Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-008-S2024/it202-milestone-1-2024/grade/jhr4

IT202-008-S2024 - [IT202] Milestone 1 2024

Submissions:

Submission Selection

1 Submission [active] 3/25/2024 7:56:37 PM

Instructions

COLLAPSE

Prereqs:

Go through each lesson from "Project Setup and SQL" to "User Login Enhancement" and follow the branching names while gathering the code
- Merge each into Milestone1 branch
- Mark the related GitHub Issues items as "done"

Implement your own custom CSS (something much different than the default "ugly" CSS given as an example)
- Consider styling all forms/inputs, data output, navigation, etc

Implement JavaScript validation on Register, Logout, and Profile (include "[Client]" in the output messages to differentiate between server-side validations)

Instructions:

Make sure you're in Milestone1 with the latest changes pulled
Ensure Milestone1 has been deployed to heroku dev
Gather the requested evidence and fill in the explanations per each prompt
Save the submission and generate the output PDF
Put the output PDF into your local repository folder
add/commit/push it to GitHub
Merge Milestone1 into dev
Locally checkout dev and pull the changes
Create and merge a pull request from dev to prod to deploy Milestone1 to prod
Upload this output PDF to Canvas

Branch name: Milestone1

Tasks: 26 Points: 10.00

User Registration (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

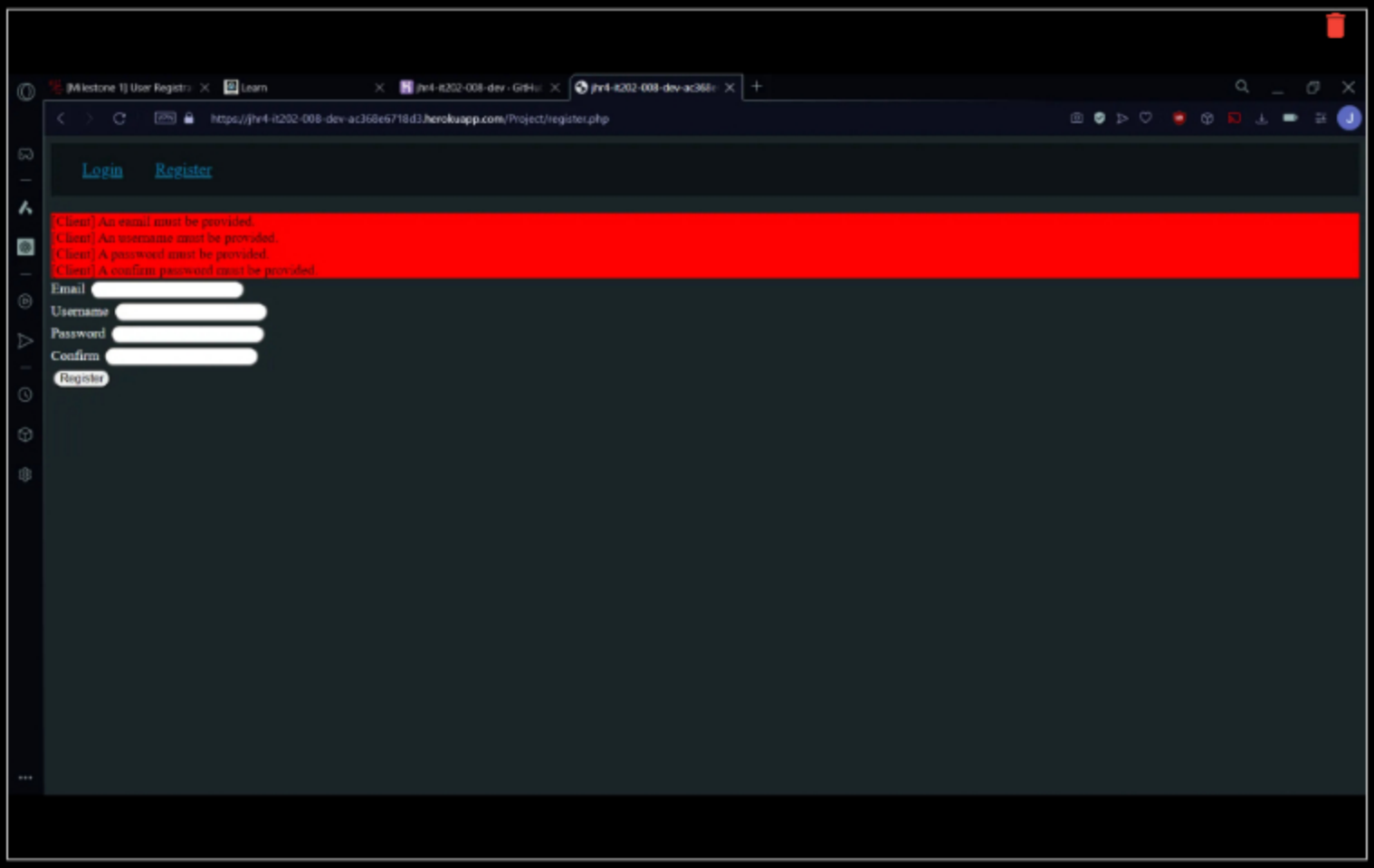
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required
#4	1	Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)
#5	1	Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)
#6	1	Demonstrate user-friendly message of new account being created

Task Screenshots:

Gallery Style: Large View

Small Medium Large



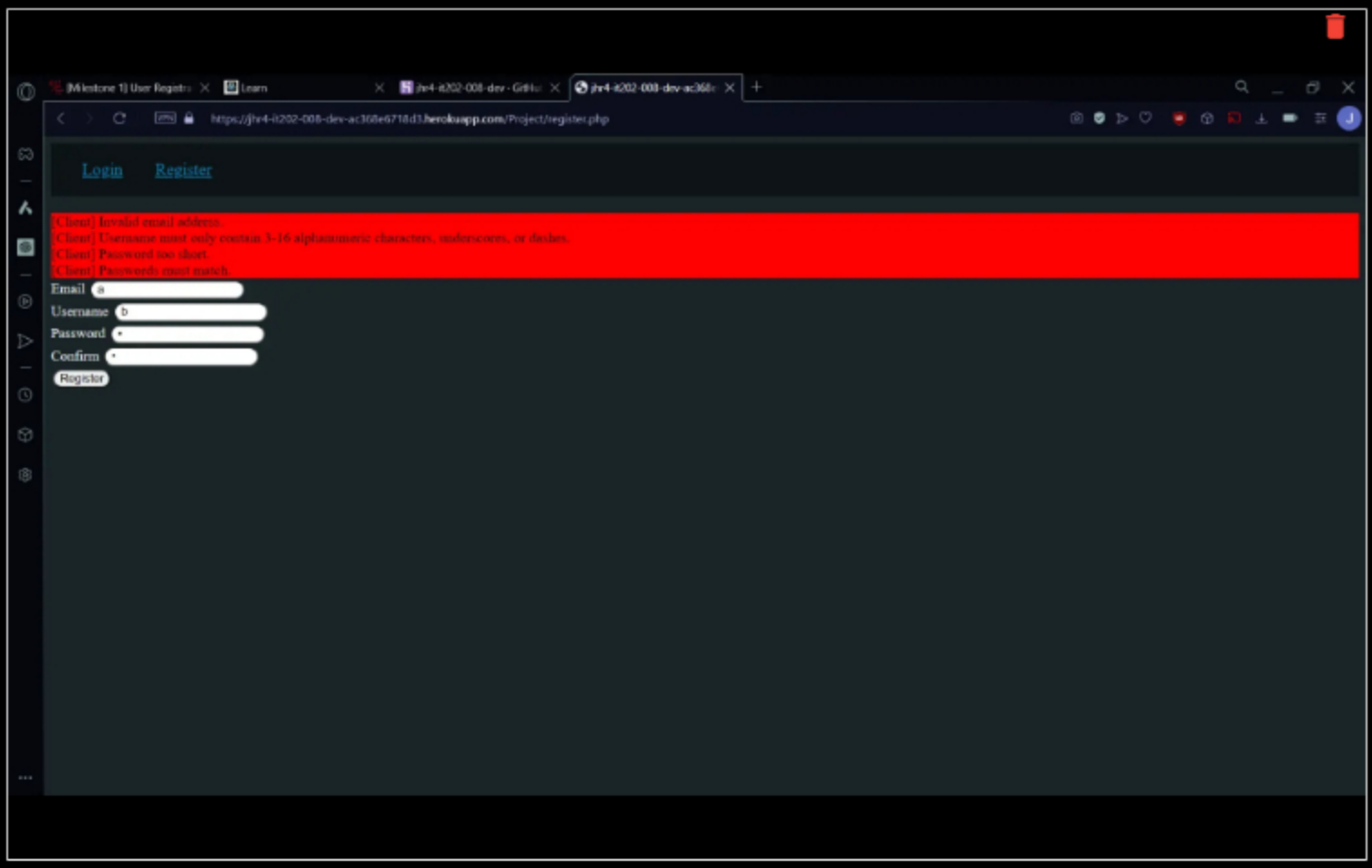
Screenshot Demonstrates Client Side (JS) Validation for if the fields are filled in or empty.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required



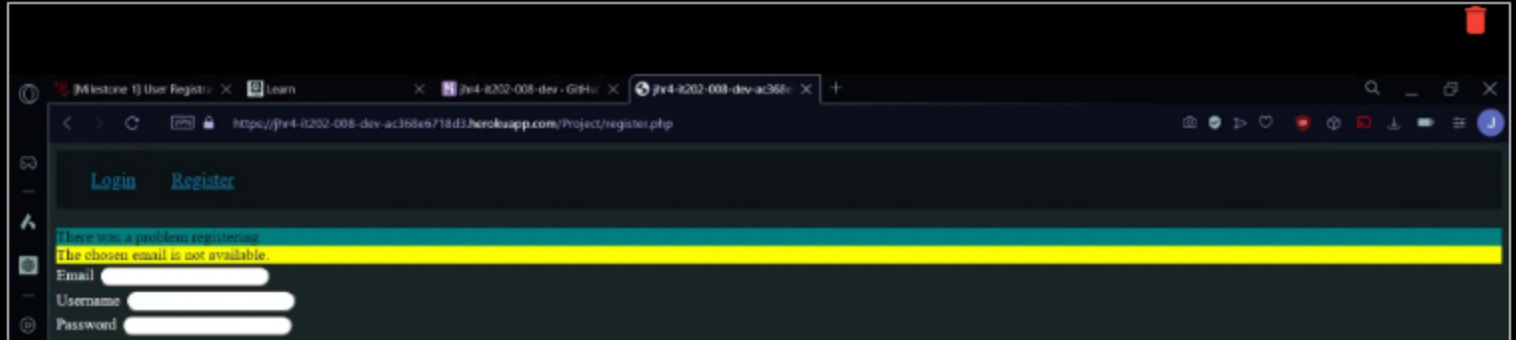
Shows JS Validation of Username, Email, and Passwords for having correct characters and size (format). Also shows the CSS.

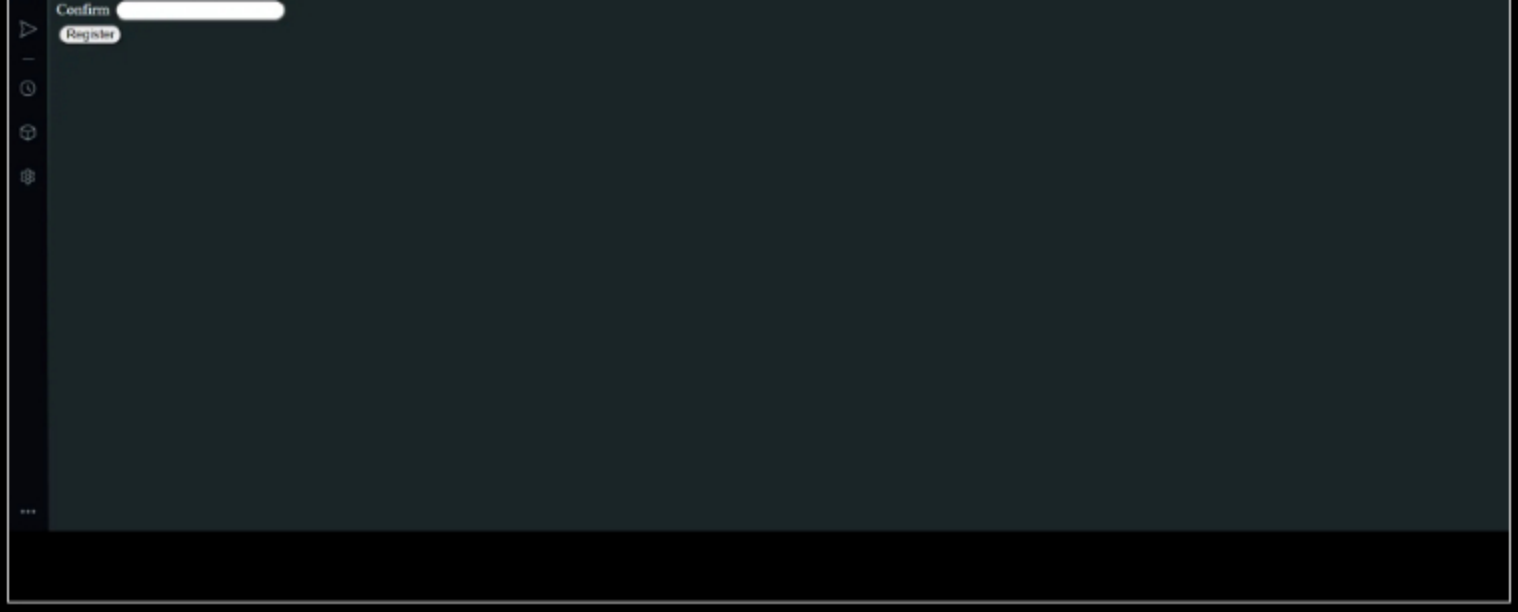
Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required





Shows email already in use message.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#4 Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)

Shows username already in use message.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#5 Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)

Shows new account created/registered message.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#6 Demonstrate user-friendly message of new account being created

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the form code

Details:
Should have appropriate input types for the fiel

Task Screenshots:

Gallery Style: Large View

Small Medium Large

jhr4-IT202-008

register.php

```php
<?php
require(__DIR__ . "/../../partials/nav.php");
reset_session();
?>
<form onsubmit="return validate(this)" method="POST">
    <div>
        <label for="email">Email</label>
        <input id="email" type="email" name="email" required/>
    </div>
    <div>
        <label for="username">Username</label>
        <input id="username" type="text" name="username" required/>
    </div>
    <div>
        <label for="pw">Password</label>
        <input type="password" id="pw" name="password" minlength="8" required/>
    </div>
    <div>
        <label for="confirm">Confirm</label>
        <input type="password" name="confirm" minlength="8" required/>
    </div>
    <input type="submit" value="Register" />
</form>
```

Form code with correct types.

COLLAPSE

Task #3 - Points: 1

Text: Screenshot of the client-side and server-side validation code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show the JavaScript validations (include any extra files related)
#2	1	Show the PHP validations (include any lib content)
#3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

Terminal Help jhr4-IT202-008

register.php

```
<script>
    function validate(form) {
        //email
        if (email === ""){
            flash("[Client] An email must be provided.", "danger");
```

```
        errorCounter++;
    }
    if (email !== "" && !/^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6})*$/.test(email)){
        flash("[Client] Invalid email address.", "danger");
        errorCounter++
    }
    //username
    if (username === ""){
        flash("[Client] An username must be provided.", "danger")
        errorCounter++
    }
    if (username !== "" && !/^[a-z0-9_-]{3,16}$/.test(username)){
        flash("[Client] Username must only contain 3-16 alphanumeric characters, underscores, or dashes.", "danger");
        errorCounter++;
    }
    //password
    if (password === ""){
        flash("[Client] A password must be provided.", "danger");
        errorCounter++;
    }
    if (confirmPassword === ""){
        flash("[Client] A confirm password must be provided.", "danger");
        errorCounter++;
    }
    if ((password !== "" && confirmPassword !== "") && (password.length < 8 || confirmPassword.length<8)){
    //tells the user if the password is to short if they attempt to change using any feild
        flash("[Client] Password too short.", "danger");
        errorCounter++;
    } <- #63-67 if ((password !== "" && confirmPassword !== "") && (password....
    if ((password !== "" && confirmPassword !== "") && confirmPassword != password){
    //only occurs if both password filled; if one feild is empty the message to fill them is already displayed so this is unnecessary
        flash("[Client] Passwords must match.", "danger");
        errorCounter++;
    } <- #68-72 if ((password !== "" && confirmPassword !== "") && confirmPas...

    if (errorCounter === 0){
        return true;
    }else {
        return false;
    }
    //jnr4 march 28, 2024
} <- #26-80 function validate(form)
```

Shows logic part of JS validation (the variables couldn't fit on ss, but they are there on top.) No extra files besides flash messages on last screenshot. Shows UCID and Date on bottom of ss.

Checklist Items (2)

#1 Show the JavaScript validations (include any extra files related)

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

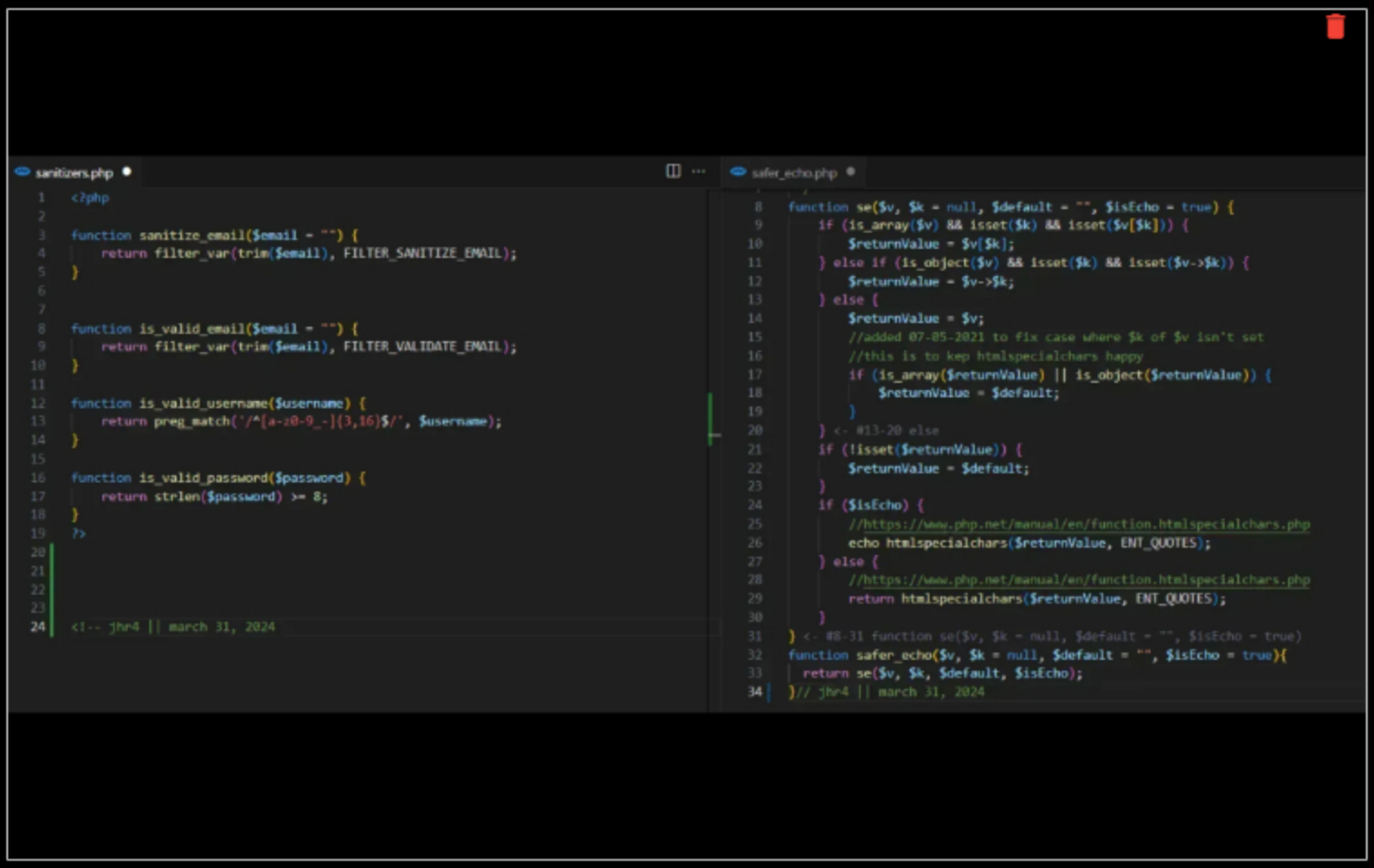
```
    //TODO 3: validate/use
$hasError = false;
    if (empty($email)) {
        flash("An email must be provided.", "danger");
        $hasError = true;}

    //sanatize removes all illegal characters
    $email = sanitize_email($email);

    //validate
    if (!is_valid_email($email)) {
        flash("Invalid email address.", "danger");
        $hasError = true;
    }
    if (!is_valid_username($username)) {
        flash("Username must only contain 3-16 alphanumeric characters, underscores, or dashes.", "danger");
        $hasError = true;
    }
    if (empty($password)) {
        flash("A password must be provided.", "danger");
        $hasError = true;
    }
    if (empty($confirm)) {
        flash("A confirm password must be provided.", "danger");
        $hasError = true;
    }
    if (!is_valid_password($password)) {
        flash("Password too short.", "danger");
        $hasError = true;
    }
    if (strlen($password) > 0 && $password !== $confirm) {
        flash("Passwords must match.", "danger");
        $hasError = true;
    }
    if(!$hasError){
        //echo "Welcome, $email";

        //TODO 4
        $hash = password_hash($password, PASSWORD_BCRYPT);          //always different output even if same password
        $db = getDB();
        $stmt = $db->prepare("INSERT INTO Users (email, password, username) VALUES(:email, :password, :username)");
        //:password & :email is placeholder;  not $varaibles because user can put in drop table as password
        try {
            $stmt->execute([":email" => $email, ":password" => $hash, ":username" => $username]);
            flash("Successfully registered!", "success");
        } catch (PDOException $e) {
            flash("There was a problem registering");
            users_check_duplicate($e->errorInfo);;
        } //jnr4 - March 28, 2024
    } <- #124-139 if(!$hasError)
} <- #84-140 if(isset($_POST["email"]) && isset($_POST["password"]) && iss...
```

Shows PHP validation code. (lib files on next screenshot) UCID & Date on Bottom.

Checklist Items (2)

#2 Show the PHP validations (include any lib content)

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)



sanitizers.php

```php
<?php

function sanitize_email($email = "") {
    return filter_var(trim($email), FILTER_SANITIZE_EMAIL);
}


function is_valid_email($email = "") {
    return filter_var(trim($email), FILTER_VALIDATE_EMAIL);
}

function is_valid_username($username) {
    return preg_match('/^[a-z0-9_-]{3,16}$/', $username);
}

function is_valid_password($password) {
    return strlen($password) >= 8;
}
?>



<!-- jhr4 || march 31, 2024
```

safer_echo.php

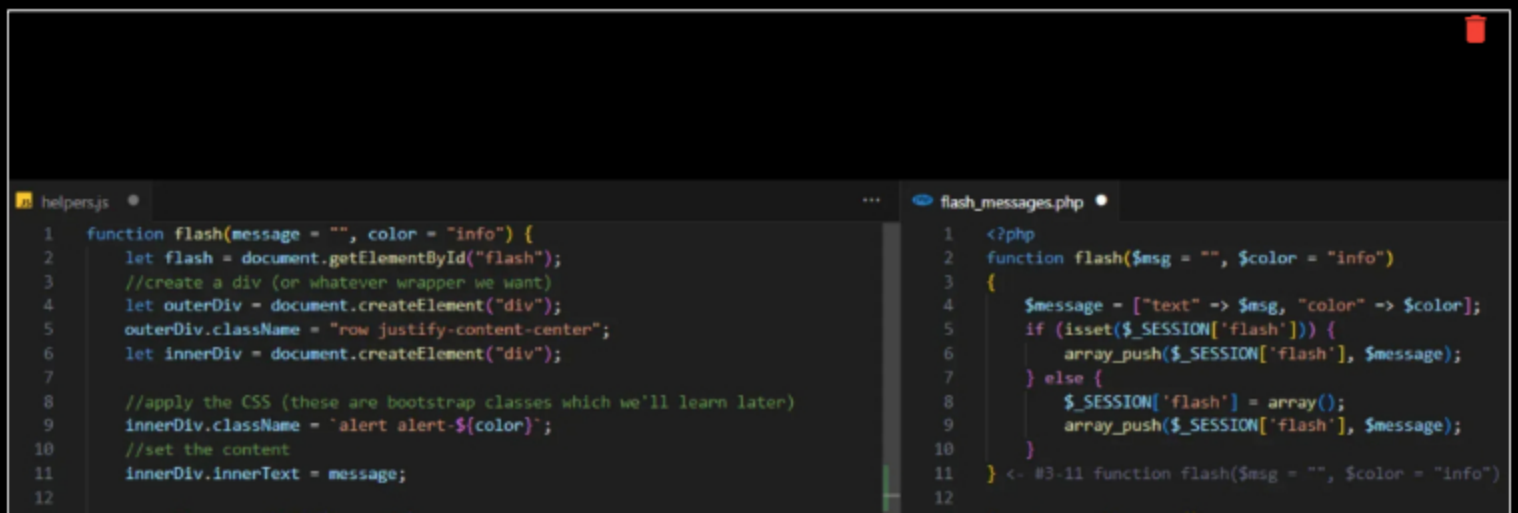
```php
function se($v, $k = null, $default = "", $isEcho = true) {
    if (is_array($v) && isset($k) && isset($v[$k])) {
        $returnValue = $v[$k];
    } else if (is_object($v) && isset($k) && isset($v->$k)) {
        $returnValue = $v->$k;
    } else {
        $returnValue = $v;
        //added 07-05-2021 to fix case where $k of $v isn't set
        //this is to kep htmlspecialchars happy
        if (is_array($returnValue) || is_object($returnValue)) {
            $returnValue = $default;
        }
    } <- #13-20 else
    if (!isset($returnValue)) {
        $returnValue = $default;
    }
    if ($isEcho) {
        //https://www.php.net/manual/en/function.htmlspecialchars.php
        echo htmlspecialchars($returnValue, ENT_QUOTES);
    } else {
        //https://www.php.net/manual/en/function.htmlspecialchars.php
        return htmlspecialchars($returnValue, ENT_QUOTES);
    }
} <- #8-31 function se($v, $k = null, $default = "", $isEcho = true)
function safer_echo($v, $k = null, $default = "", $isEcho = true){
  return se($v, $k, $default, $isEcho);
}// jhr4 || march 31, 2024
```

Lib content for PHP validation (safer echo to assign variable for the user inputs), and sanitizers (valid username email and password check with regex and character length. also the built in PHP sanitizer) UCID & Date on Bottom.

Checklist Items (2)

#2 Show the PHP validations (include any lib content)

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)



helpers.js

```javascript
function flash(message = "", color = "info") {
    let flash = document.getElementById("flash");
    //create a div (or whatever wrapper we want)
    let outerDiv = document.createElement("div");
    outerDiv.className = "row justify-content-center";
    let innerDiv = document.createElement("div");

    //apply the CSS (these are bootstrap classes which we'll learn later)
    innerDiv.className = `alert alert-${color}`;
    //set the content
    innerDiv.innerText = message;

```

flash_messages.php

```php
<?php
function flash($msg = "", $color = "info")
{
    $message = ["text" => $msg, "color" => $color];
    if (isset($_SESSION['flash'])) {
        array_push($_SESSION['flash'], $message);
    } else {
        $_SESSION['flash'] = array();
        array_push($_SESSION['flash'], $message);
    }
} <- #3-11 function flash($msg = "", $color = "info")

```

```
    outerDiv.appendChild(innerDiv);
    //add the element to the DOM (if we don't it merely exists in memory)
    flash.appendChild(outerDiv);
} <- #1-16 function flash(message = "", color = "info")

// jhr4 || march 31, 2024
```

```
function getMessages()
{
    if (isset($_SESSION['flash'])) {
        $flashes = $_SESSION['flash'];
        $_SESSION['flash'] = array();
        return $flashes;
    } <- #15-19 if (isset($_SESSION['flash']))
    return array();
} <- #14-21 function getMessages()

// jhr4 || march 31, 2024
```

Flash message files for PHP and JS utilized to display the user friendly messages. UCID & Date on Bottom.

Checklist Items (3)

#1 Show the JavaScript validations (include any extra files related)

#2 Show the PHP validations (include any lib content)

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

COLLAPSE

Task #4 - Points: 1

Text: Screenshot of the Users table with a valid user entry

Checklist

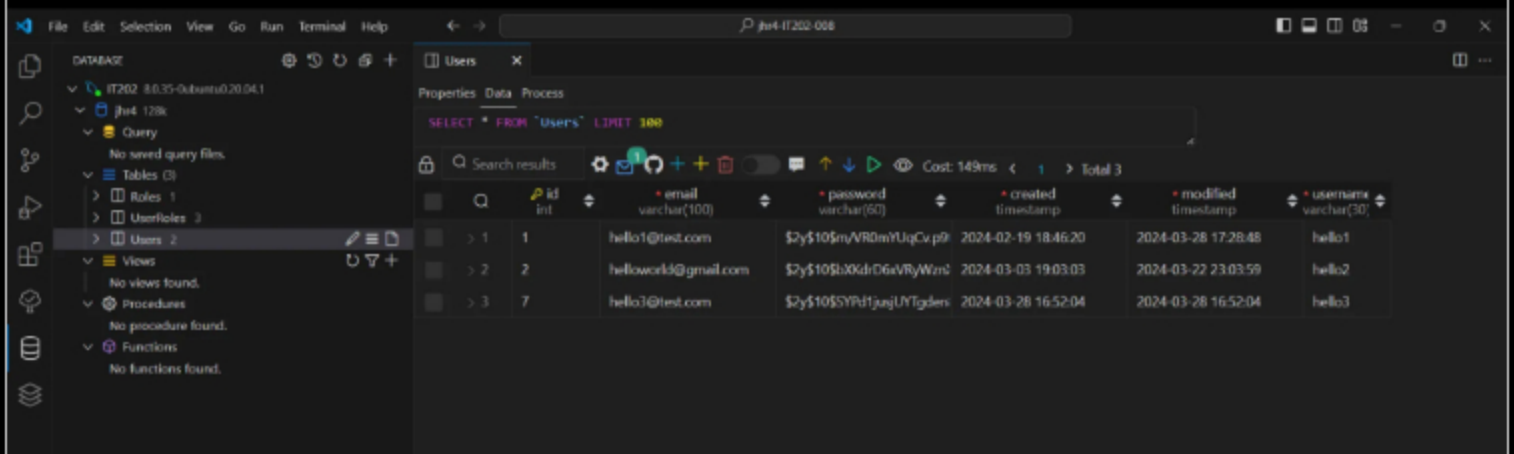
*The checkboxes are for your own tracking

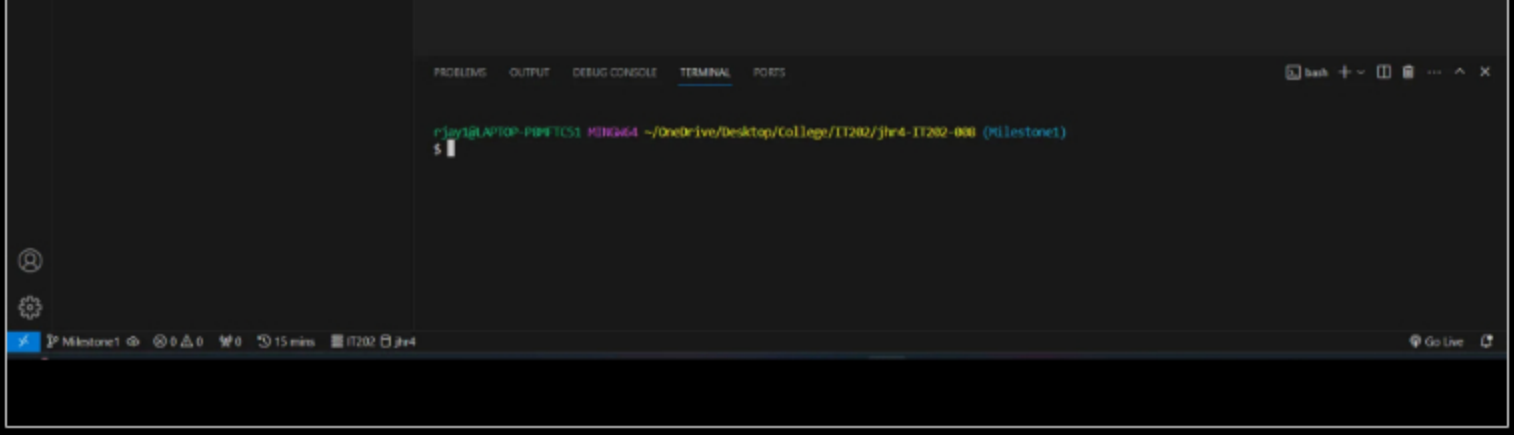
#	Points	Details
#1	1	Password should be hashed
#2	1	Should have email, password, username (unique), created, modified, and id fields
#3	1	Ensure left panel or database name is present (should contain your ucid)

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Shows Users table with email, password, username (unique), created, mortified, and id fields. The passwords are hashed. UCID visible to left.

Checklist Items (3)

#1 Password should be hashed

#2 Should have email, password, username (unique), created, modified, and id fields

#3 Ensure left panel or database name is present (should contain your ucid)

COLLAPSE

Task #5 - Points: 1

Text: Explain the registration logic in a step-by-step manner from when the page loads to when the data is saved to the DB

Details:
Don't just show code, translate things to plain English

Response:

The registration page has three stages basically: client side validation (JS & HTML), server side validation (PHP), and the mySQL part with a database.

When the page's html/css loads it displays a form with email, username, password, and confirm password. The JS and HTML validates this by checking: (1) if they are empty or not (via 'required' attribute in html and the 'return onsubmit="validate()"' which calls the JS function and returns true or false based on if the fields are === "" or not. (2) if the basic requirements/format is met via html utilizing the type="email" and "length" for password/username. Also JS validates it using regex to check if the character requirements are met or not for fields like the username and email. (3) The js code also checks if the confirm password is the same as the original.

However, as this is on the client side and can be modified by the client, there is also server side validation (php) that does the same thing as the JS, but in php there's also sanitization (removing illegal characters).

Furthermore, if the validation is met with no errors it moves onto hashing the password and calling the database to store the user information into the Users table. If there's an error during this like email has been taken, this error message is displayed to the user. Another thing to note is that in the SQL the user inputs aren't directly stored, but

rather placeholders are used to avoid SQL injections.

If any validation fails or there's an error an user friendly message is displayed (using the created flash function) for the user to try again.

COLLAPSE

Task #6 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/Jhr-4/jhr4-IT202-008/pull/20

User Login (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Include correct data where username is used to login
#4	1	Include correct data where email is used to login
#5	1	Show success login message
#6	1	Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)
#7	1	Demonstrate user-friendly message of when an account doesn't exist
#8	1	Demonstrate user-friendly message of when password doesn't match what's in the DB
#9	1	Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)
#10	1	Demonstrate session data being set (captured from server logs)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

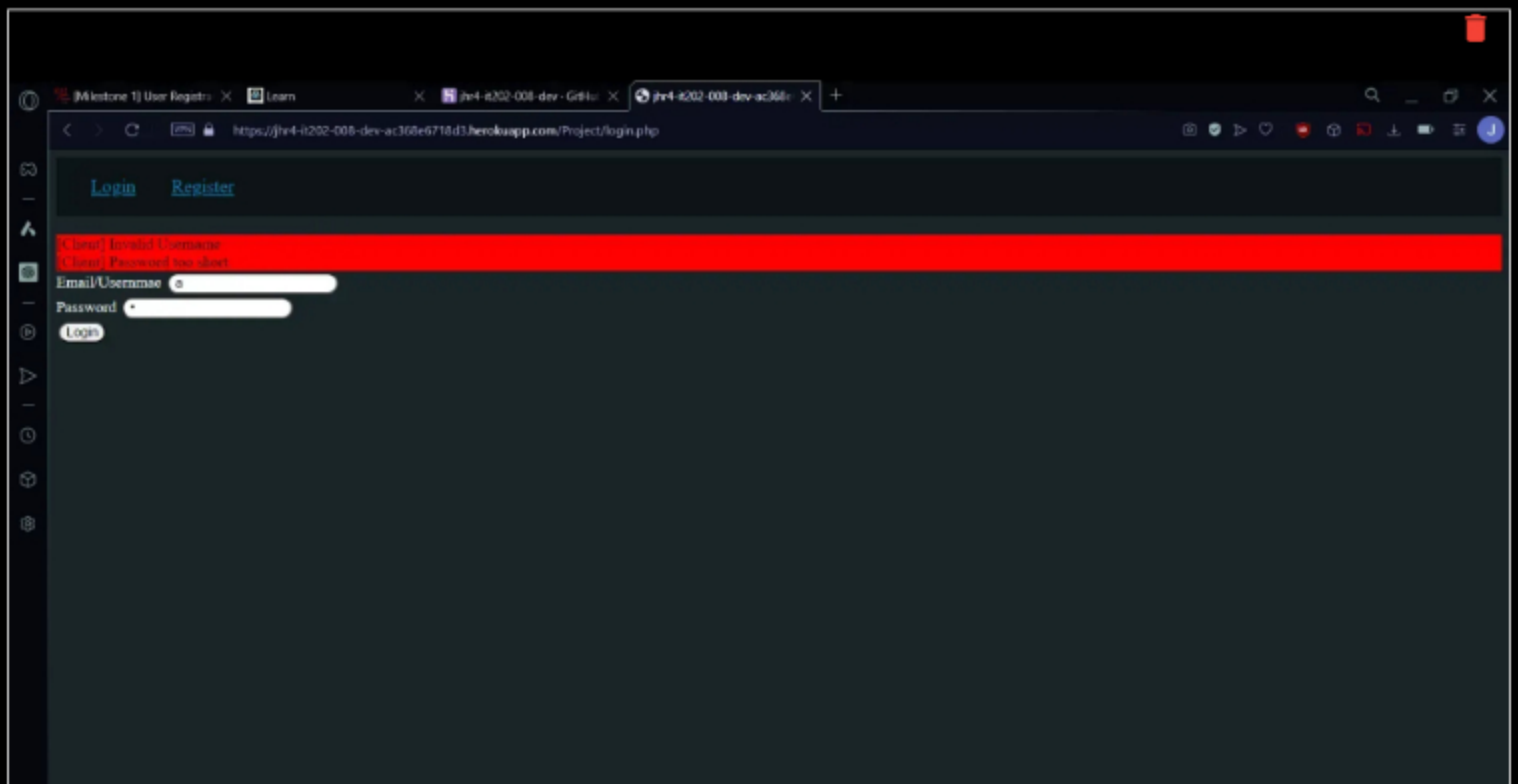
Shows JS validation of each field being required. Shows CSS styling that's on every page too. The dev URL is on top too.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)



Shows JS validation of username and password format. Username error is shown when no @ is present.

Checklist Items (2)

#3 Include correct data where username is used to login

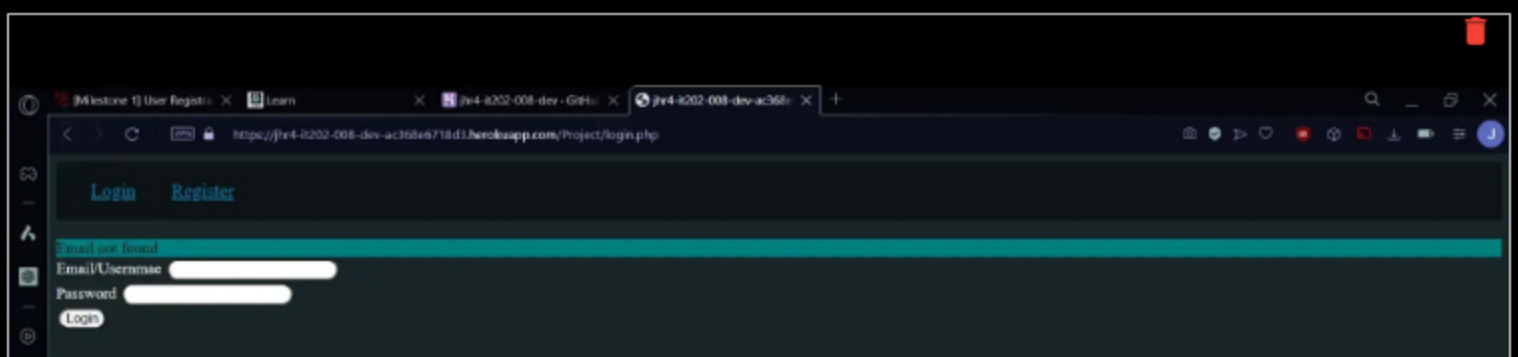
#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)

Shows JS validation of email (if it includes an "@" symbol).

Checklist Items (2)

#4 Include correct data where email is used to login

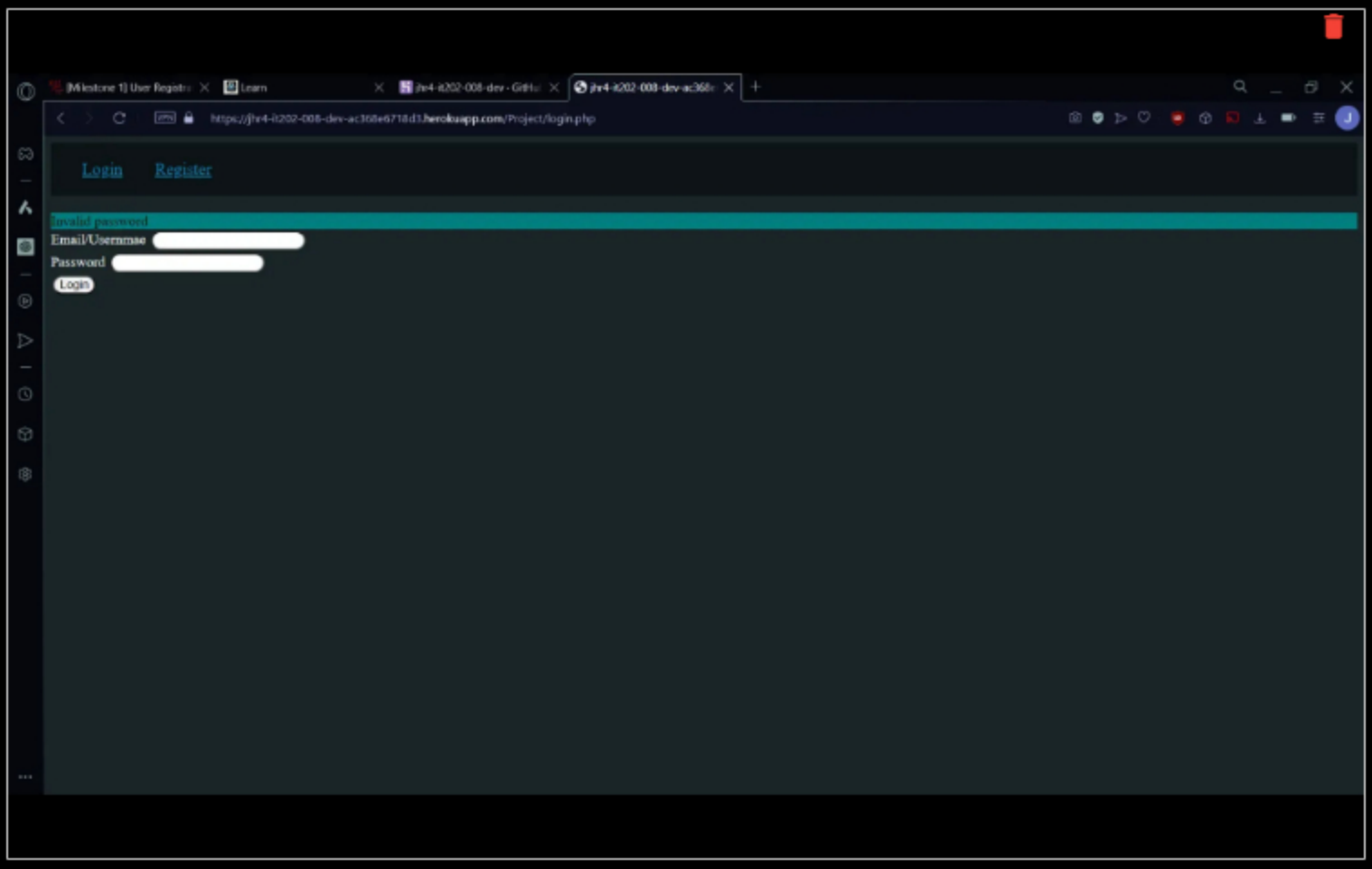
#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)



Shows message of when account doesn't exist or not found.

Checklist Items (1)

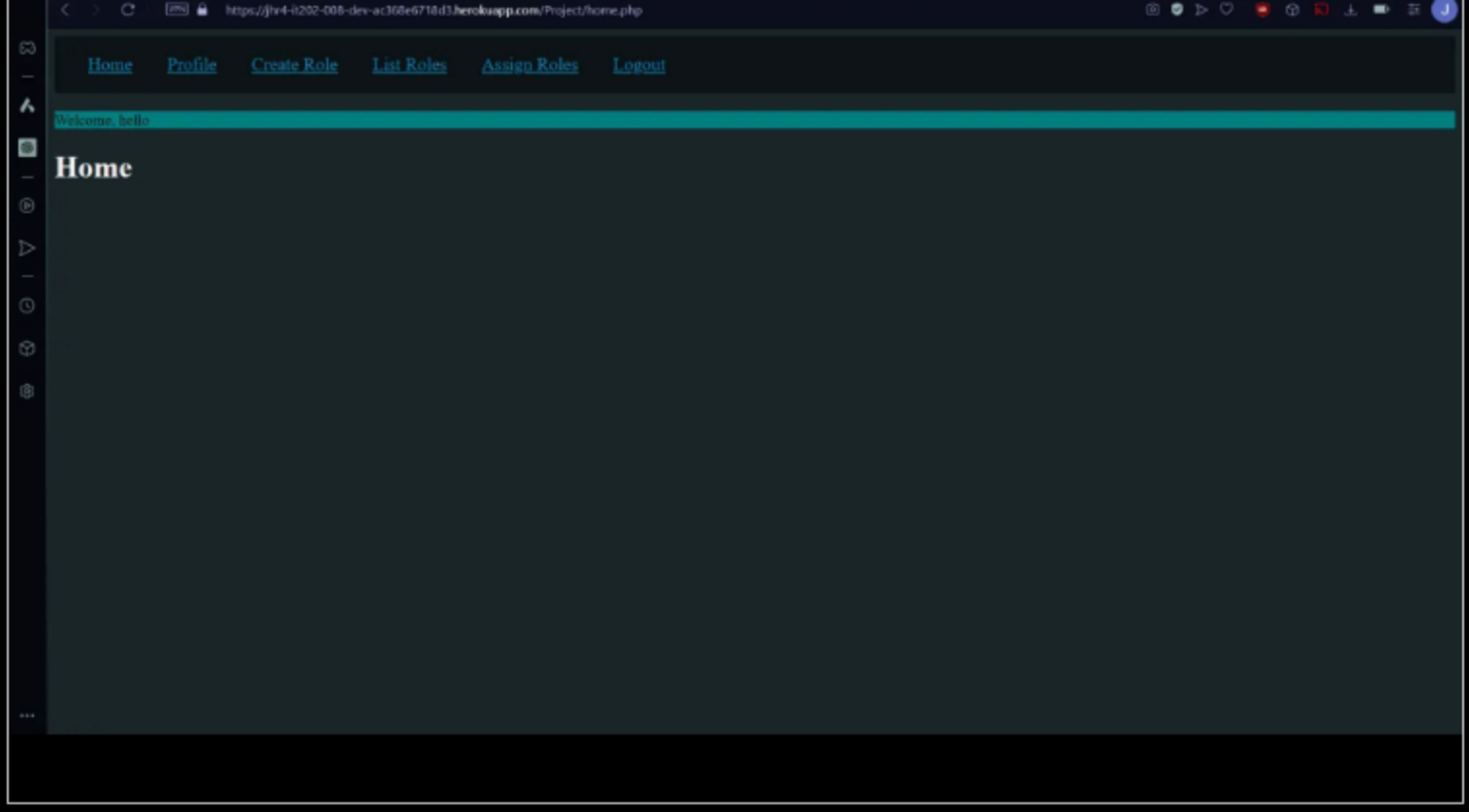
#7 Demonstrate user-friendly message of when an account doesn't exist



Shows message of password not matching what's in DB.

Checklist Items (1)

#8 Demonstrate user-friendly message of when password doesn't match what's in the DB

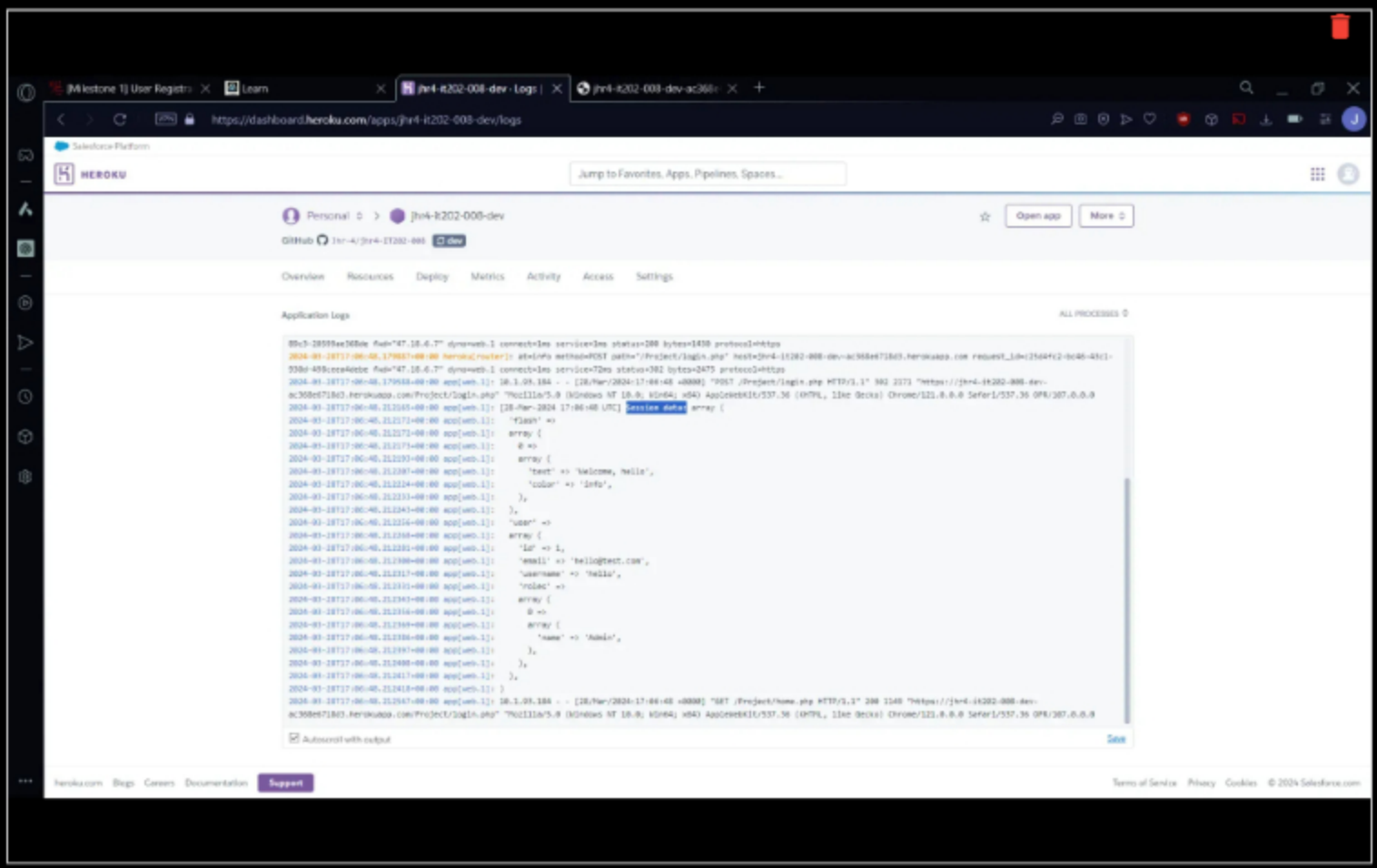


Shows successful login message in home page after login via a "welcome."

Checklist Items (2)

#5 Show success login message

#9 Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)



Shows session data being captured in Herkou server logs.

Checklist Items (1)

#10 Demonstrate session data being set (captured from server logs)

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the form code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Should have proper input types for the fields (note in the caption if you're using two fields for Username/Email or a combined field)
#2	1	Show JavaScript validations (include any extra files related)
#3	1	Show PHP validations (include any lib content)
#4	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
4   <form onsubmit="return validate(this)" method="POST">
5       <div>
6           <label for="email">Email/Usernmae</label>
7           <input type="text" name="email" required/>
8       </div>
9       <div>
10          <label for="pw">Password</label>
11          <input type="password" id="pw" name="password" minlength="8" required/>
12      </div>
13      <input type="submit" value="Login"/>
14  </form>
15      <!-- jhr4 || march 28, 2024-->
16  <script>
```

Shows HTML form code with two fields: Email/Username and Password. The type for the first field is text because it can be username too. ucid/date on bottom.

Checklist Items (2)

#1 Should have proper input types for the fields (note in the caption if you're using two fields for Username/Email or a combined field)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

```
<!-- jhr4 || march 28, 2024-->
<script>
    function validate(form) {
        //TODO 1: implement JavaScript validation
        //ensure it returns false for an error and true for success
        let password = form.password.value;
        let email_user = form.email.value;

        let isValid = true;

        //email or username
        if (email_user === ""){ //if empty it must be provided
                flash("[Client] Email/username must be provided.", "danger");
                isValid = false;
        } else if (/@/.test(email_user)){ //not empty from earlier & if has @ -> check email regex conditions
            if ( !/^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6})*$/.test(email_user)){
                flash("[Client] Invalid Email", "danger");
                isValid = false;
            }
        } else if (!/^[a-z0-9_-]{3,16}$/.test(email_user)){ //not empty from earlier & no @email -> check username regex conditions
                flash("[Client] Invalid Username", "danger"); //user should already know format...
                isValid = false;
        }

        //password
        if (password === ""){
            flash("[Client] A password must be provided.", "danger");
            isValid = false;
        }
        if (password !=="" && password.length < 8){
            flash("[Client] Password too short.", "danger");//will potentially be changed to just "Invalid Password." User should know correct length.
            isValid = false;
        }

        return isValid;
    } <- #17-50 function validate(form)
</script>
```

JS validation code. ucid/date on top.

Checklist Items (2)

#2 Show JavaScript validations (include any extra files related)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

```
//TODO 3
$hasError = false;
if (empty($email)) {
    flash("Email must not be empty");
    $hasError = true;
}
if (str_contains($email, "@")) {
//sanitize
//$email = filter_var($email, FILTER_SANITIZE_EMAIL);
$email = sanitize_email($email);

//validate
/*if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    flash("Invalid email address");
    $hasError = true;
}*/
    if (!is_valid_email($email)) {
        flash("Invalid email address");
        $hasError = true;
    }
} else {
    if (!is_valid_username($email)) {
        flash("Invalid username");
```

```php
            $hasError = true;
        }
    } <- #77-82 else

    if (empty($password)) {
        flash("password must not be empty");
        $hasError = true;
    }
    if (strlen($password) < 8) {
        flash("Password too short");
        $hasError = true;
    }    //jhr4 || march 28, 2024
    if (!$hasError) {
```

PHP validation (format & existence) UCID and date on bottom.

Checklist Items (0)

login.php

```php
$db = getDB();
$stmt = $db->prepare("SELECT id, email, username, password from Users where email = :email or username = :email");
try {
    $r = $stmt->execute([":email" => $email]);
    if ($r) {
        $user = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($user) {
            $hash = $user["password"];
            unset($user["password"]);
            if (password_verify($password, $hash)) {
                $_SESSION["user"] = $user;
                try {
                    //lookup potential roles
                    $stmt = $db->prepare("SELECT Roles.name FROM Roles
                JOIN UserRoles on Roles.id = UserRoles.role_id
                where UserRoles.user_id = :user_id and Roles.is_active = 1 and UserRoles.is_active = 1");
                    $stmt->execute([":user_id" => $user["id"]]);
                    $roles = $stmt->fetchAll(PDO::FETCH_ASSOC); //fetch all since we'll want multiple
                } catch (Exception $e) {
                    error_log(var_export($e, true));
                }
                //save roles or empty array
                if (isset($roles)) {
                    $_SESSION["user"]["roles"] = $roles; //at least 1 role
                } else {
                    $_SESSION["user"]["roles"] = []; //no roles
                }
                flash("Welcome, ". get_username());
                die(header("Location: home.php"));
            } else {
                flash("Invalid password");
            }
        } else {
            flash("Email not found");
        }
    } <- #100-129 if ($user)
} catch (Exception $e) {
    flash("<pre>" . var_export($e, true) . "</pre>"); //jhr4 || march 28, 2024
```

Validation with database (making sure email/username exists in db and password matches to one stored in DB) ucid & date on bottom

Checklist Items (2)

#3 Show PHP validations (include any lib content)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

sanitizers.php

```php
<?php

function sanitize_email($email = "") {
    return filter_var(trim($email), FILTER_SANITIZE_EMAIL);
}
```

safer_echo.php

```php
function se($v, $k = null, $default = "", $isEcho = true) {
    if (is_array($v) && isset($k) && isset($v[$k])) {
        $returnValue = $v[$k];
    } else if (is_object($v) && isset($k) && isset($v->$k)) {
        $returnValue = $v->$k;
```

```php
function is_valid_email($email = "") {
    return filter_var(trim($email), FILTER_VALIDATE_EMAIL);
}

function is_valid_username($username) {
    return preg_match('/^[a-z0-9_-]{3,16}$/', $username);
}

function is_valid_password($password) {
    return strlen($password) >= 8;
}
?>



<!-- jhr4 || march 31, 2024
```

```php
    } else {
        $returnValue = $v;
        //added 07-05-2021 to fix case where $k of $v isn't set
        //this is to kep htmlspecialchars happy
        if (is_array($returnValue) || is_object($returnValue)) {
            $returnValue = $default;
        }
    } <- #13-20 else
    if (!isset($returnValue)) {
        $returnValue = $default;
    }
    if ($isEcho) {
        //https://www.php.net/manual/en/function.htmlspecialchars.php
        echo htmlspecialchars($returnValue, ENT_QUOTES);
    } else {
        //https://www.php.net/manual/en/function.htmlspecialchars.php
        return htmlspecialchars($returnValue, ENT_QUOTES);
    }
} <- #8-31 function se($v, $k = null, $default = "", $isEcho = true)
function safer_echo($v, $k = null, $default = "", $isEcho = true){
  return se($v, $k, $default, $isEcho);
}// jhr4 || march 31, 2024
```

Same as register. user_helpers isn't shown as that's not apart of validation it occurs after login and storing data into session. Lib content for PHP validation (safer echo to assign variable for the user inputs), and sanitizers (valid username email and password check with regex and character length. also the built in PHP sanitizer) UCID & Date on Bottom.

Checklist Items (2)

#3 Show PHP validations (include any lib content)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

helpers.js

```javascript
function flash(message = "", color = "info") {
    let flash = document.getElementById("flash");
    //create a div (or whatever wrapper we want)
    let outerDiv = document.createElement("div");
    outerDiv.className = "row justify-content-center";
    let innerDiv = document.createElement("div");

    //apply the CSS (these are bootstrap classes which we'll learn later)
    innerDiv.className = `alert alert-${color}`;
    //set the content
    innerDiv.innerText = message;

    outerDiv.appendChild(innerDiv);
    //add the element to the DOM (if we don't it merely exists in memory)
    flash.appendChild(outerDiv);
} <- #1-16 function flash(message = "", color = "info")

// jhr4 || march 31, 2024
```

flash_messages.php

```php
<?php
function flash($msg = "", $color = "info")
{
    $message = ["text" => $msg, "color" => $color];
    if (isset($_SESSION['flash'])) {
        array_push($_SESSION['flash'], $message);
    } else {
        $_SESSION['flash'] = array();
        array_push($_SESSION['flash'], $message);
    }
} <- #3-11 function flash($msg = "", $color = "info")

function getMessages()
{
    if (isset($_SESSION['flash'])) {
        $flashes = $_SESSION['flash'];
        $_SESSION['flash'] = array();
        return $flashes;
    } <- #15-19 if (isset($_SESSION['flash']))
    return array();
} <- #14-21 function getMessages()

// jhr4 || march 31, 2024
```

Same as register. Flash message files for PHP and JS utilized to display the user friendly messages. UCID & Date on Bottom

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Explain the login logic step-by-step from when the page loads to when the data is fetched from the DB and stored in the session

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Don't just show code, translate things to plain English
#2	1	Explain how the session works and why/how it's used

Response:

The login logic is similar to the register page for the validation, however this time the email/username field is validated based on the condition of if it has an @ or not which tells us if the user is trying to put an email or username. After this the validation is basically the same and was explained earlier.

After again similar to before, we access the database's Users table. This time however, we select the email or username (based on what the user put) by using a placeholder of the input. We also select the password corresponding to that email/user. If there's no data for that email we return email not found. After this we check the password using PHP's "password_verify" method using the hashed password from the db (which is unset) and the users typed in password.

If the user was found and the passwords match, the users data array from the db is stored in the session and also the roles information is gathered of the user and stored in the session. The session was actually loaded when the person loaded up the page as the navbar is required. The navbar utilizes the session_start() method in php. This looks for the PHPSESSID cookie on the client side and sees if that session exists on the server side if there is it will load it and if not it will create a new session. The cookie is also created with the navbar being loaded and has a domain & path of only the project website meaning it only exists on the website and path and not others. Basically it means the user data is transferred across pages in this website. This is important as the session is a way to verify that the user has logged in allowing the user to use other pages that should be locked for someone that's not logged in like home or profile. The idea of roles being in the session is also important as it allows for other pages to be unlocked like admin ones if they have the role for it.

In the end if everything is successful the login page's script is killed (with die) and user is redirected to home page.
If any validation fails or there's an error an user friendly message is displayed (using the created flash function) for the user to try again.

COLLAPSE

Task #4 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/Jhr-4/jhr4-IT202-008/pull/21/

User Logout (1 pt.)
COLLAPSE

COLLAPSE

Task #1 - Points: 1
Text: Capture the following screenshots

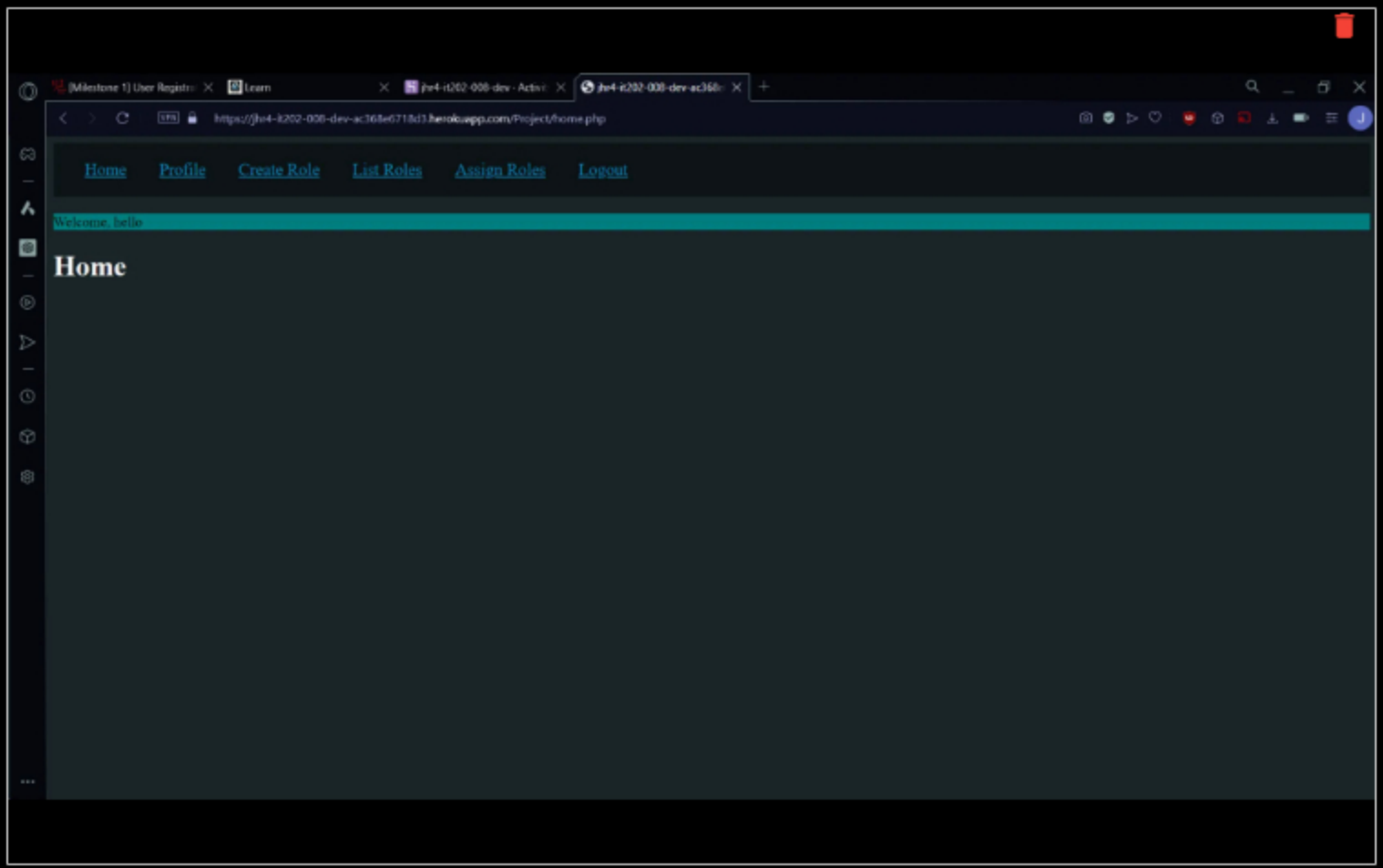
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Screenshot of the navigation when logged in (site)
#2	1	Screenshot of the redirect to login with the user-friendly logged-out message (site)
#3	1	Screenshot of the logout-related code showing the session is destroyed (code). Ensure ucid/date comment is present.

Task Screenshots:

Gallery Style: Large View

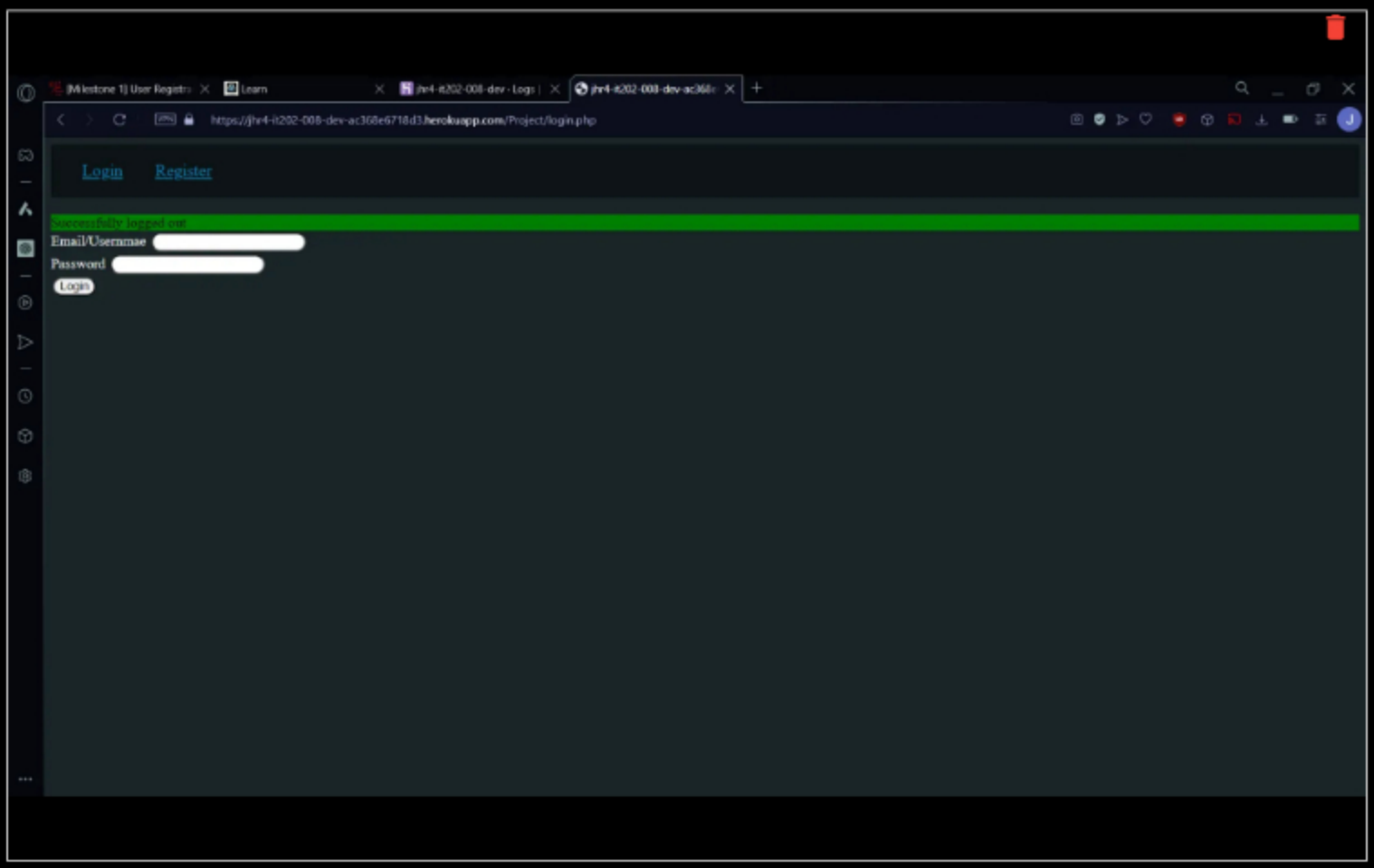
Small Medium Large



Shows nav bar of admin account login.

Checklist Items (1)

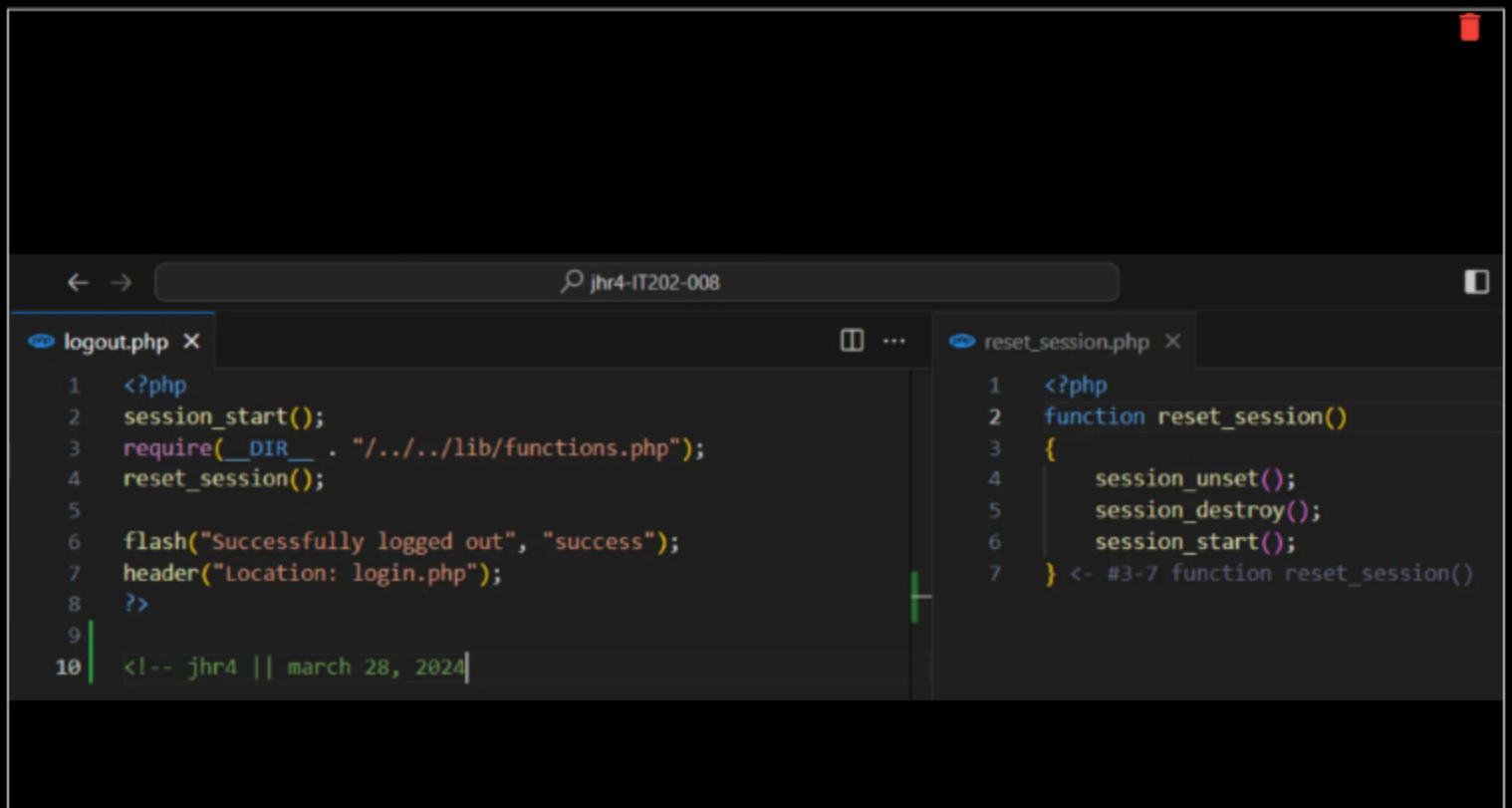
#1 Screenshot of the navigation when logged in (site)



Shows Logout message and being redirected to Login page.

Checklist Items (1)

#2 Screenshot of the redirect to login with the user-friendly logged-out message (site)



Shows how session is destroyed with the function reset_session() which is also shown on the right. ucid / date on bottom.

Checklist Items (1)

#3 Screenshot of the logout-related code showing the session is destroyed (code). Ensure ucid/date comment is present.

COLLAPSE

Task #2 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/Jhr-4/jhr4-IT202-008/pull/21

Basic Security Rules and Roles (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Authentication Screenshots

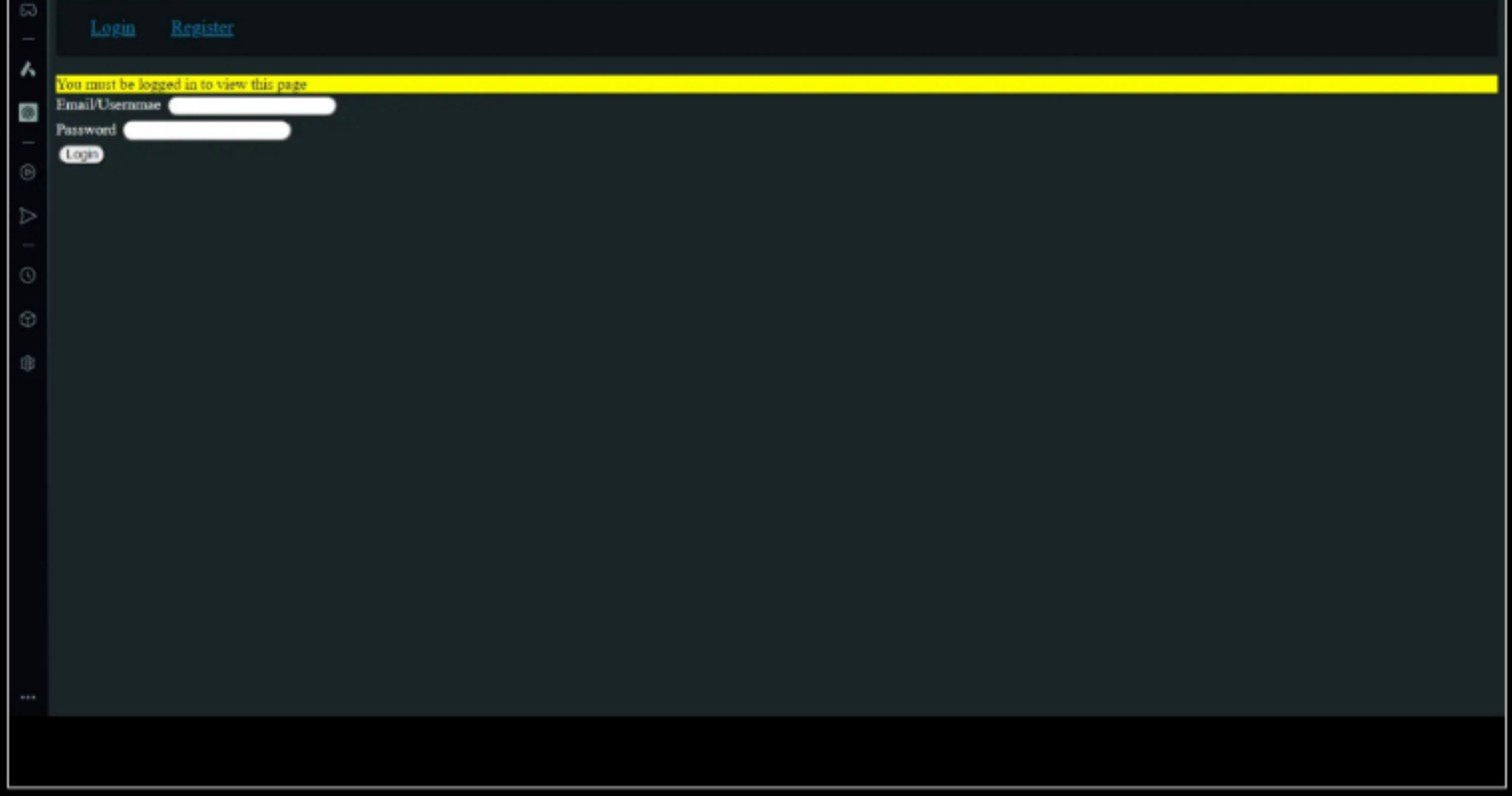
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Screenshot of the function that checks if a user is logged in
#2	1	Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on
#3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
#4	1	Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Shows message of logged out user trying to access login protected page.

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out

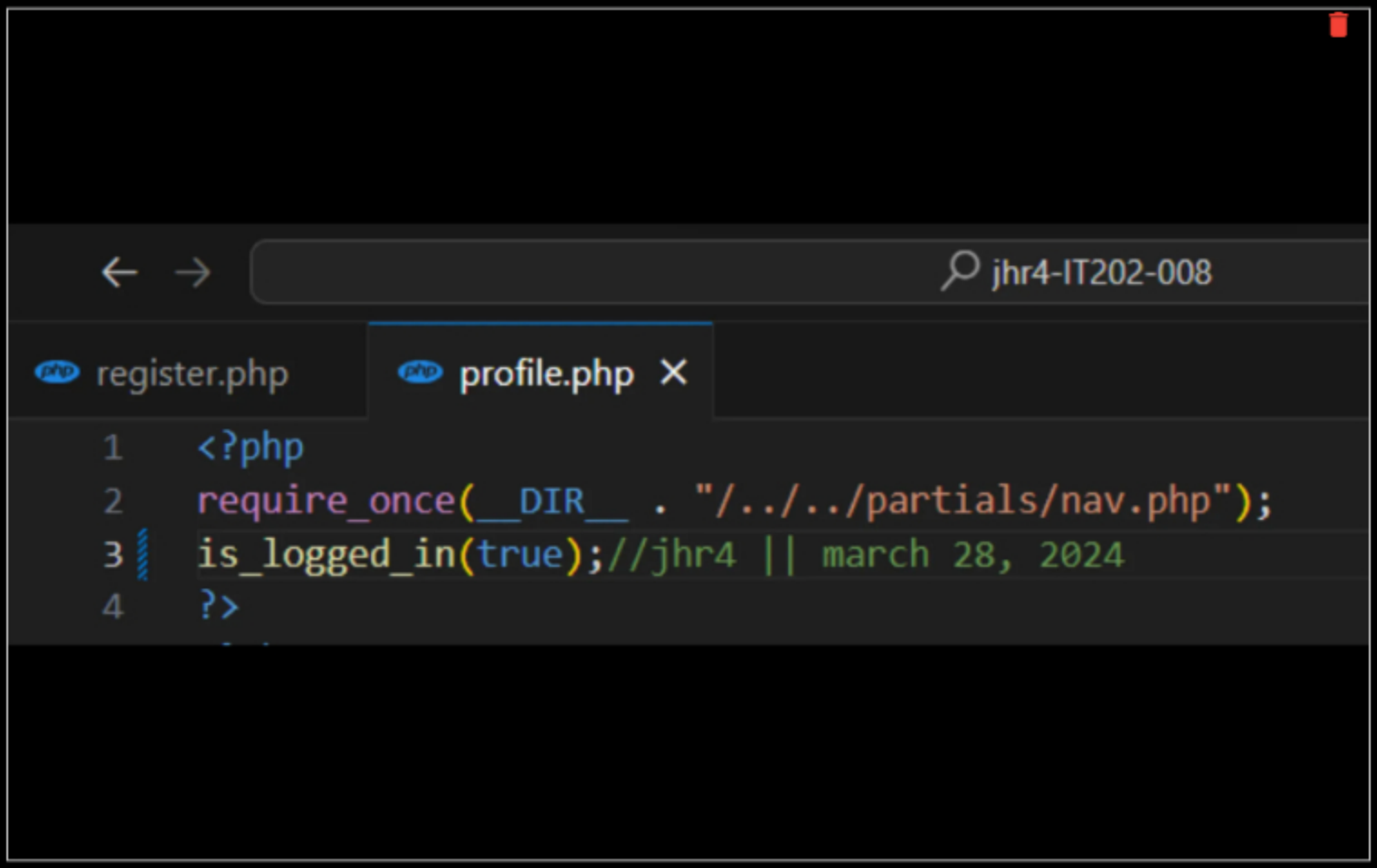
```php
function is_logged_in($redirect = false, $destination = "login.php")
{
    $isLoggedIn = isset($_SESSION["user"]);
    if ($redirect && !$isLoggedIn) {
        //if this triggers, the calling script won't receive a reply since die()/exit() terminates it
        flash("You must be logged in to view this page", "warning");
        die(header("Location: $destination"));
    } <- #10-14 if ($redirect && !$isLoggedIn)
    return $isLoggedIn;
} //jhr4 || march 28, 2024 <- #8-16 function is_logged_in($redirect = false, $destination = "logi...
```

Shows function to check if user is logged in. UCID & date on bottom.

Checklist Items (2)

#1 Screenshot of the function that checks if a user is logged in

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)



Shows login check function being used in profile page. The function directly used on: profile.php and home.php. It's also used in the has_role() function so its also used on: assign_roles.php, create_role.php, and list_roles.php. UCID & Date on Bottom

Checklist Items (2)

#2 Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

COLLAPSE

Task #2 - Points: 1

Text: Authorization Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Screenshot of the function that checks for a specific role
#2	1	Screenshot of the role check function being used. Also caption what pages it's used on
#3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
		Demonstrate the user-friendly message of trying to manually access a role-protected

#4	1	page while being logged out

Task Screenshots:

Gallery Style: Large View

Small Medium Large

Shows message of user trying to access a role protected page while being logged out.

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a role-protected page while being logged out

```php
function has_role($role) {
    if (is_logged_in() && isset($_SESSION["user"]["roles"])) {
        foreach ($_SESSION["user"]["roles"] as $r) {
            if ($r["name"] === $role) {
                return true;
            }
        } <- #19-23 foreach ($_SESSION["user"]["roles"] as $r)
    } <- #18-24 if (is_logged_in() && isset($_SESSION["user"]["r
    return false;
```

```
26  } //jhr4 || march 28, 2024 <- #17-26 function has_role($role)
```

Shows function to check for a specific role in user_helpers.php file. ucid date on bottom.

Checklist Items (2)

#1 Screenshot of the function that checks for a specific role

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

assign_roles.php | user_helpers.php

```php
<?php
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: $BASE_PATH" . "/home.php"));
}//jhr4 march 28, 2024
```

Shows role check function being used. It's used on: assign_roles.php, create_role.php, and list_roles.php UCID/date on bottom

Checklist Items (2)

#2 Screenshot of the role check function being used. Also caption what pages it's used on

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

COLLAPSE

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

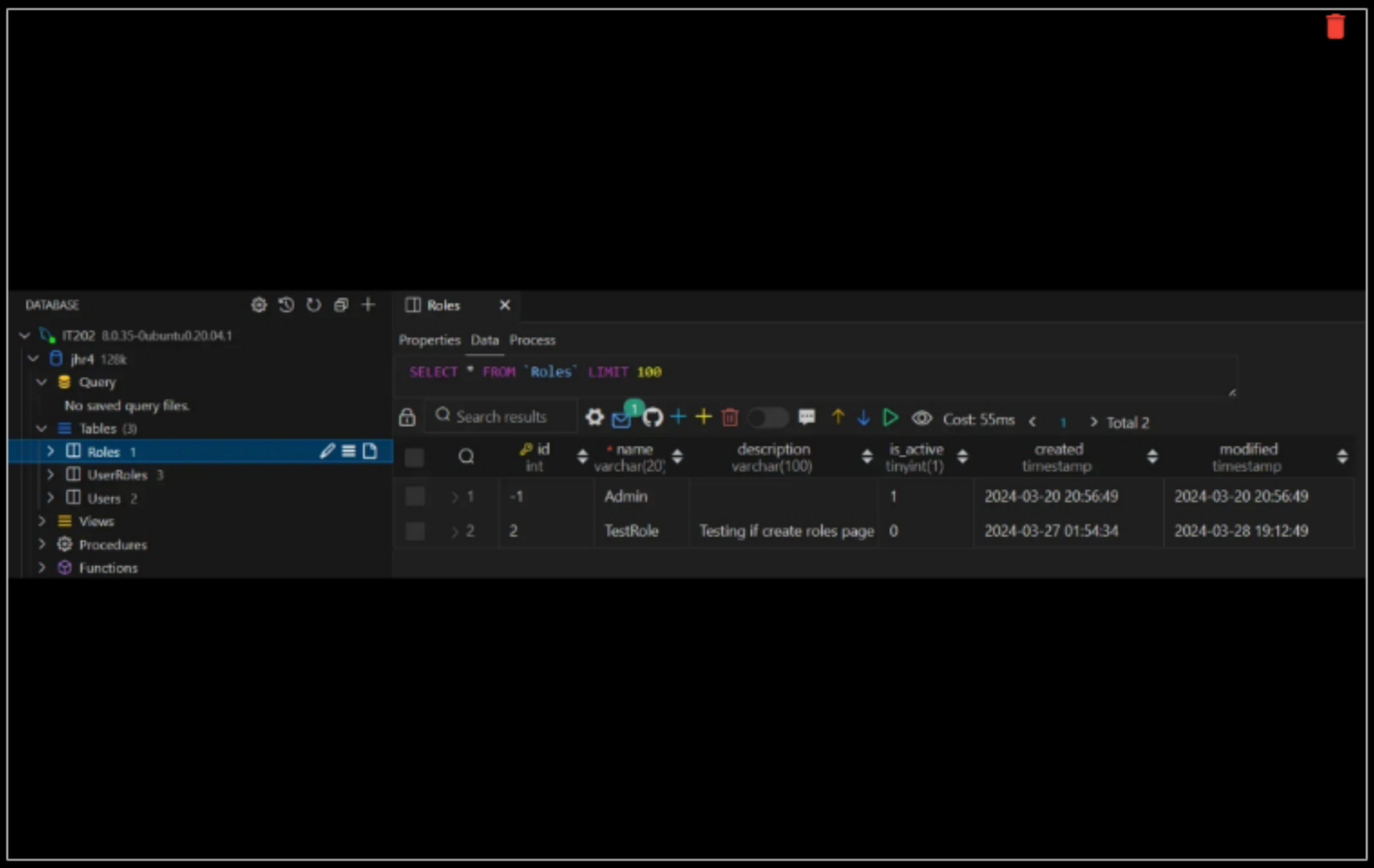
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	At least one valid and enabled User->Role reference (UserRoles table)
#2	1	UserRoles Table should have id, user_id, role_id, is_active, created, and modified columns
#3	1	Roles Table should have id, name, description, is_active, modified, and created columns
#4	1	At least one valid and enabled Role (Roles table)
#5	1	Ensure left panel or database name is present in each table screenshot (should contain your ucid)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



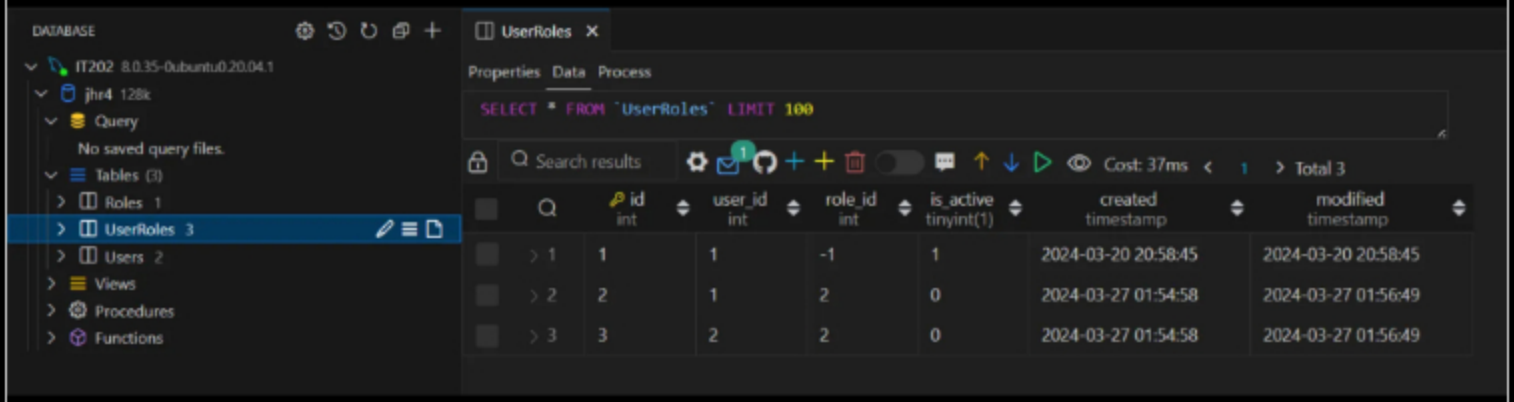
Shows roles table with admin role enabled and TestRole disabled. All 6 of these columns are present: id, name, description, is_active, modified, and created. UCID on top left.

Checklist Items (3)

#3 Roles Table should have id, name, description, is_active, modified, and created columns

#4 At least one valid and enabled Role (Roles table)

#5 Ensure left panel or database name is present in each table screenshot (should contain your ucid)



User Roles table with user_id 1 with active admin role (-1). Table contains all these columns: id, user_id, role_id, is_active, created, and modified columns. UCID on top left.

Checklist Items (3)

#1 At least one valid and enabled User->Role reference (UserRoles table)

#2 UserRoles Table should have id, user_id, role_id, is_active, created, and modified columns

#5 Ensure left panel or database name is present in each table screenshot (should contain your ucid)

COLLAPSE

Task #4 - Points: 1

Text: Explain how Roles and UserRoles tables work in conjunction with the Users table

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	What's the purpose of the UserRoles table?
#2	1	How does Roles.is_active differ from UserRoles.is_active?

Response:

The Roles table is just a list of roles in which each role name and id is unique. The UserRoles table is a combination of the Users table and Roles table in which users are given roles by utilizing their user_id and basically matching it with the role_id from the Roles table. Basically Roles table establishes the roles while UserRoles assigns the roles.

Roles.is_active for the actual role while UserRoles.is_active is for the user. If the Roles.is_active 1 it's active and it

Roles.is_active for the actual role while UserRoles.is_active is for the user. If the Roles.is_active 1 it's active and it works, but if it's 0 it's basically disabled. For UserRoles.is_active its per user so if a role is toggled for a user they don't have the role. This system basically allows roles to be "deleted" from users or the whole website. "deleted" basically just means disabled as the actual role is never removed just toggled active or inactive.

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/Jhr-4/jhr4-IT202-008/pull/27

COLLAPSE

User Profile (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: View Profile Website Page

Checklist

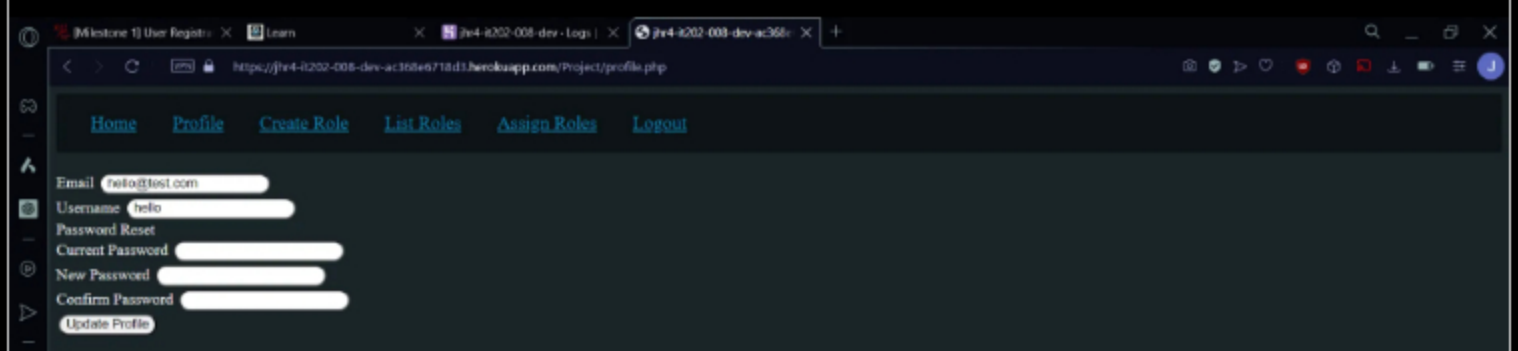
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Show the profile form correctly populated on page load (username, email)
#4	1	Should have the following fields: username, email, current password, new password, confirm password (or similar)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Shows profile page with decent CSS of inputs being rounded. There are 5 fields: email, username, current password, new password, confirm password. The username and email fields are prefilled on load. CSS will be made better when adding in Bootstraps next milestone.

Checklist Items (4)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#3 Show the profile form correctly populated on page load (username, email)

#4 Should have the following fields: username, email, current password, new password, confirm password (or similar)

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic step-by-step of how the data is loaded and populated when the profile page is visited

Details:
Don't just show code, translate things to plain English

Response:

When the profile page is visited the data that is loaded/populated is the email and username. This is done through php which utilizes the get_user_email() and get_username() function which placing it ad the value for the fields. These functions work by getting the username and email from the Session which is stored if the person is logged in.

The session is set when the user visits the website and logs in which then places the username email and roles of the user in the session which is loaded in every page thru the nav bar.

Task #3 - Points: 1

^COLLAPSE ^

Text: Edit Profile Website Page

Checklist

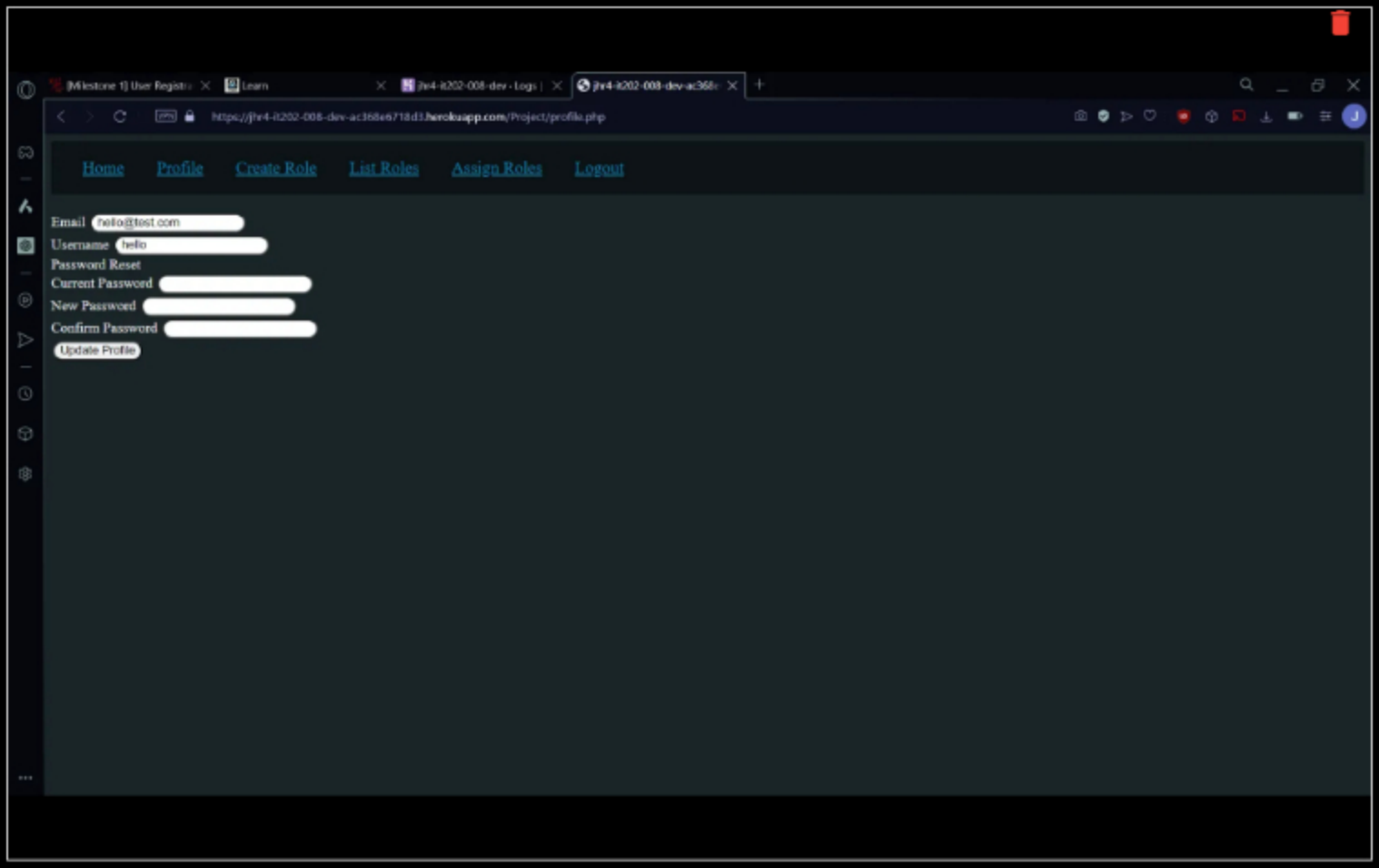
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Demonstrate with before and after of a username change (including success message)
#4	1	Demonstrate with a before and after of an email change (including success message)
#5	1	Demonstrate the success message of updating password
#6	1	Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)
#7	1	Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Before image of profile page with loaded username & email. With CSS styling and Heroku dev link on top.

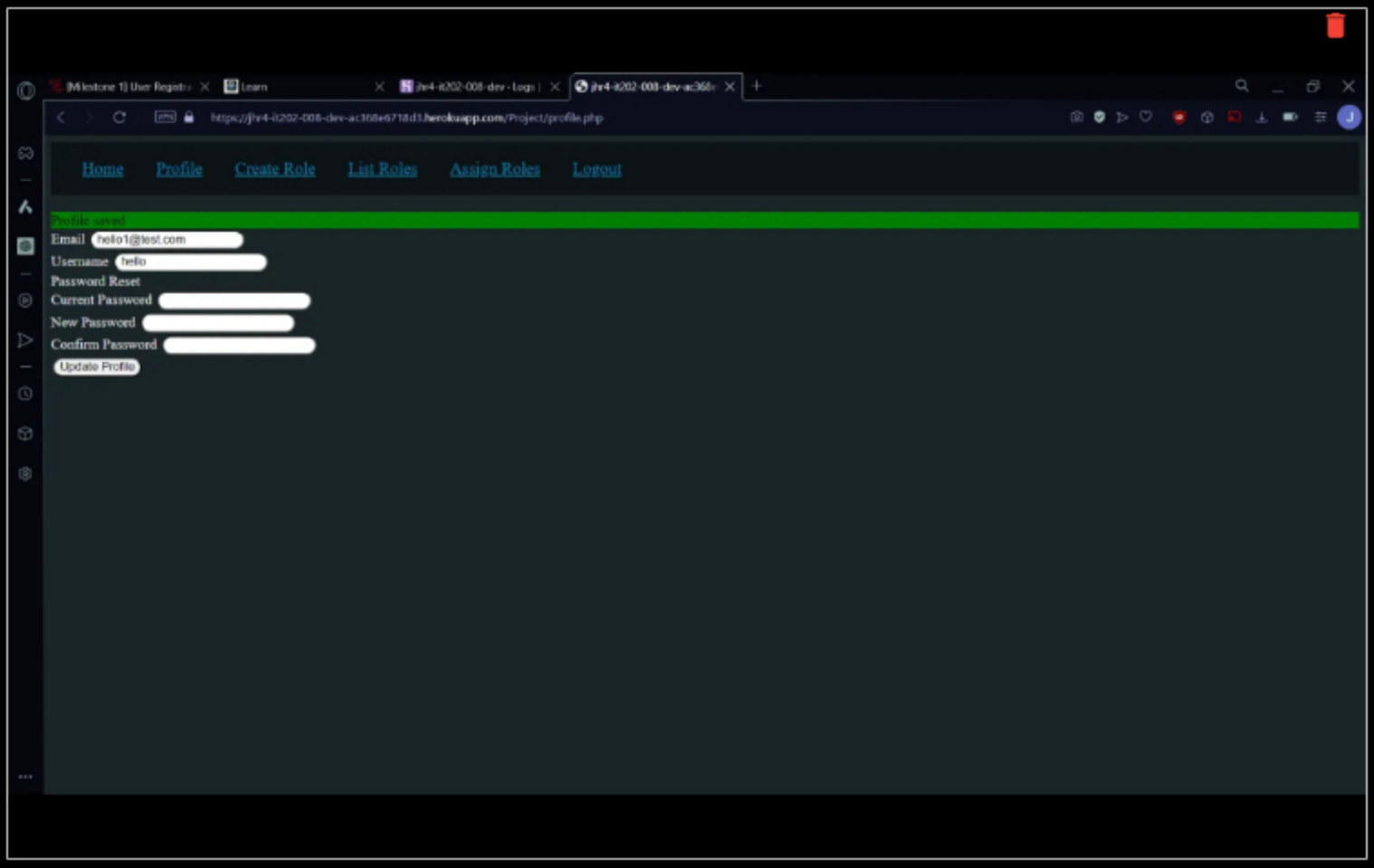
Checklist Items (4)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#3 Demonstrate with before and after of a username change (including success message)

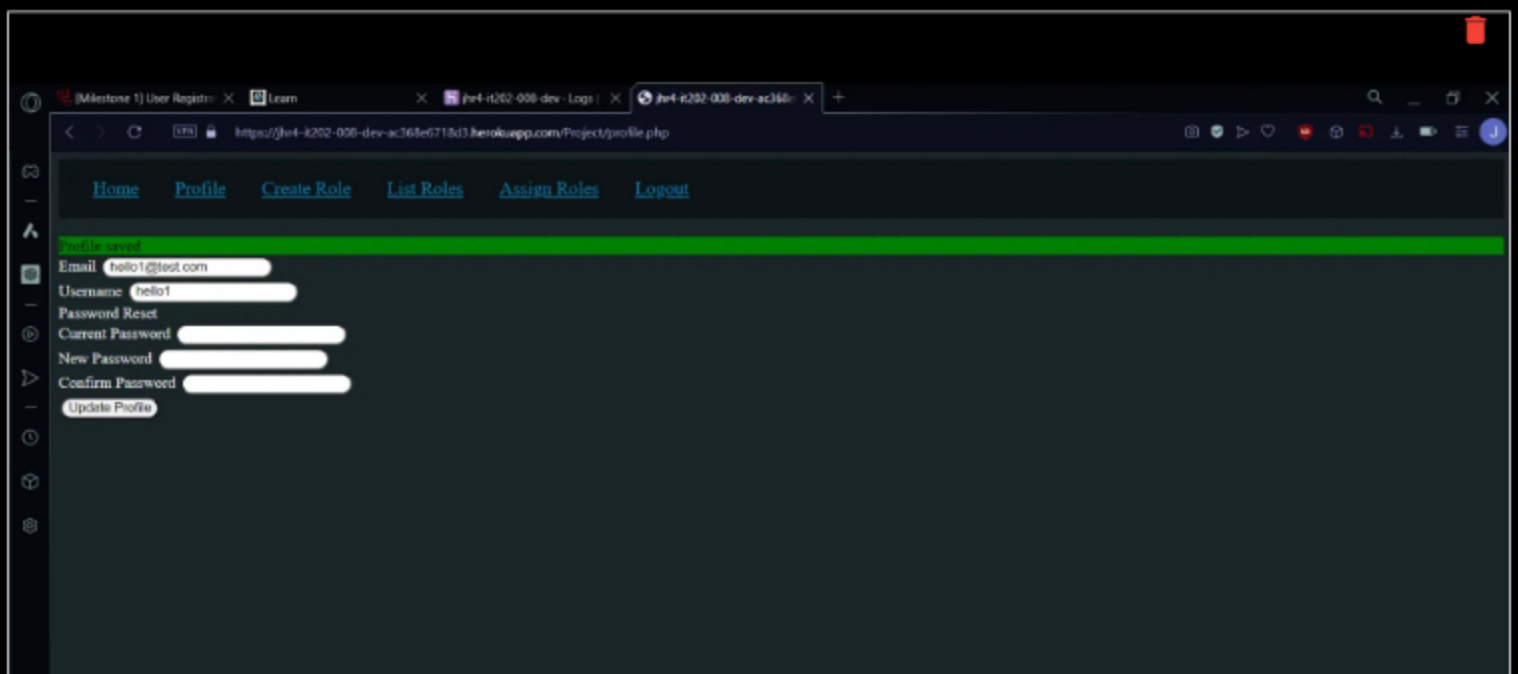
#4 Demonstrate with a before and after of an email change (including success message)



After screenshot of email being changed, success message, "profile saved."

Checklist Items (1)

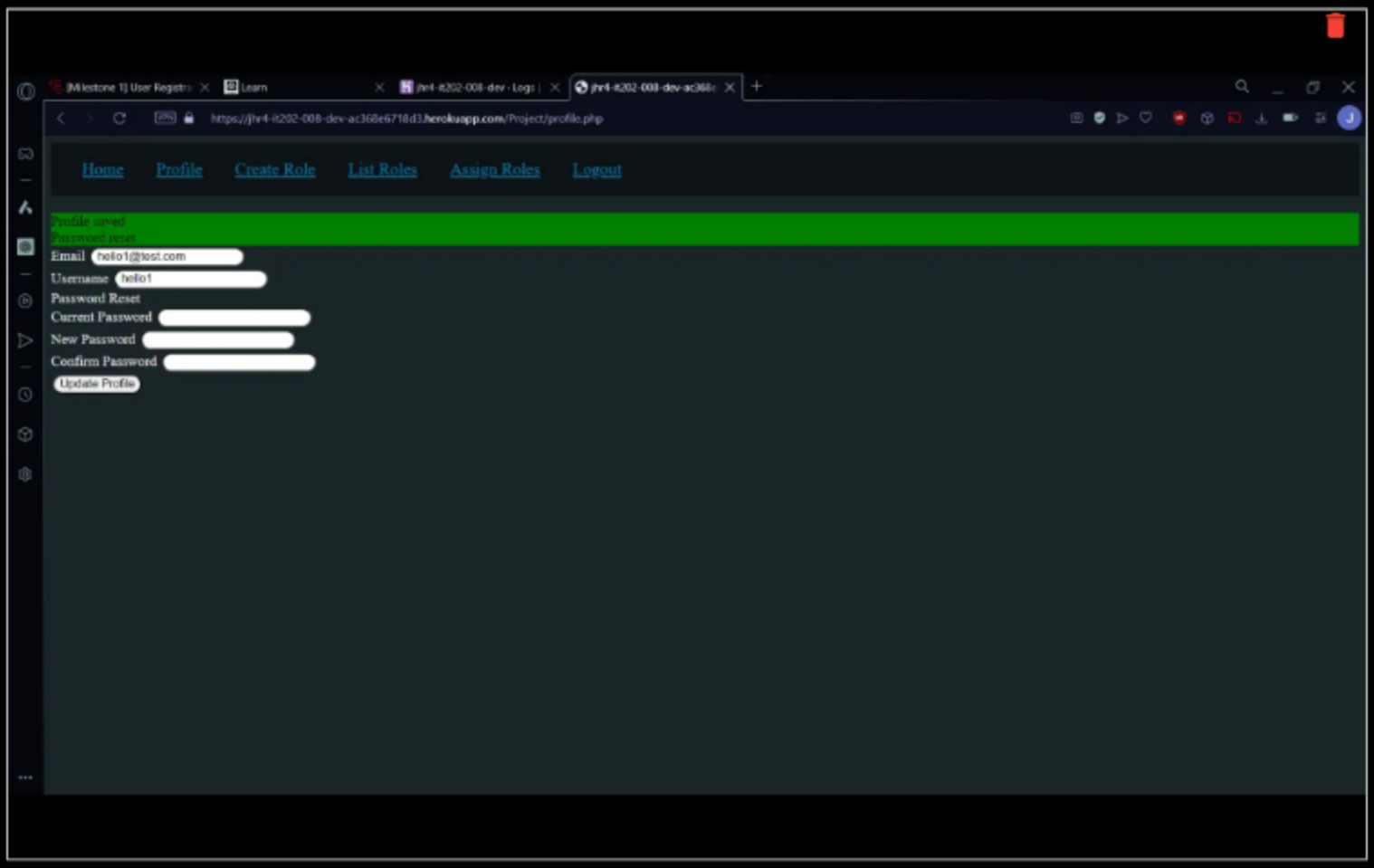
#4 Demonstrate with a before and after of an email change (including success message)



After screenshot of username being changed success image, "profile saved."

Checklist Items (1)

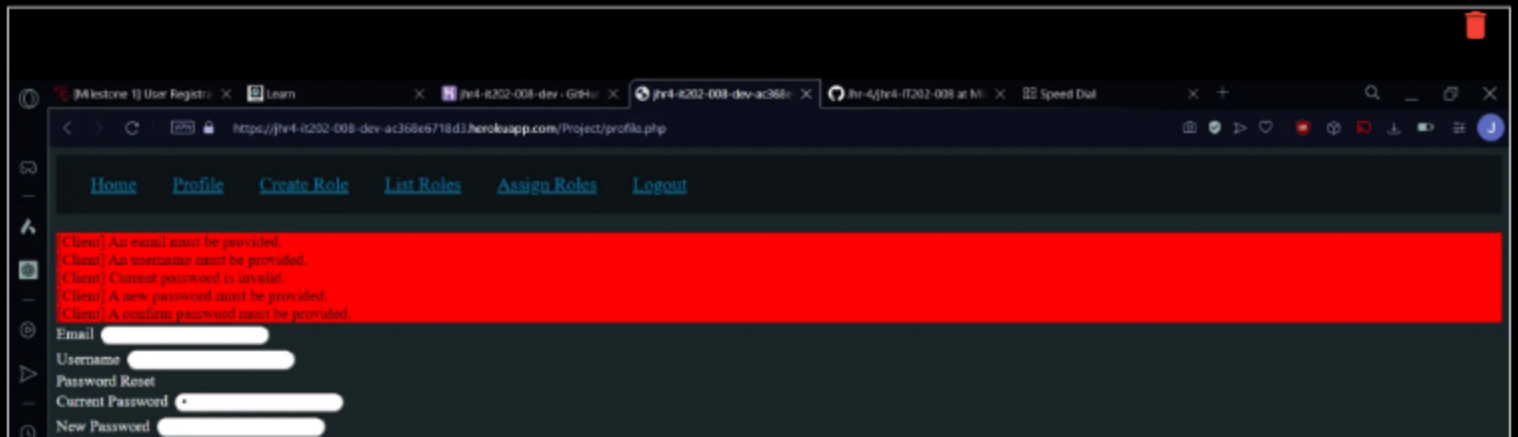
#3 Demonstrate with before and after of a username change (including success message)

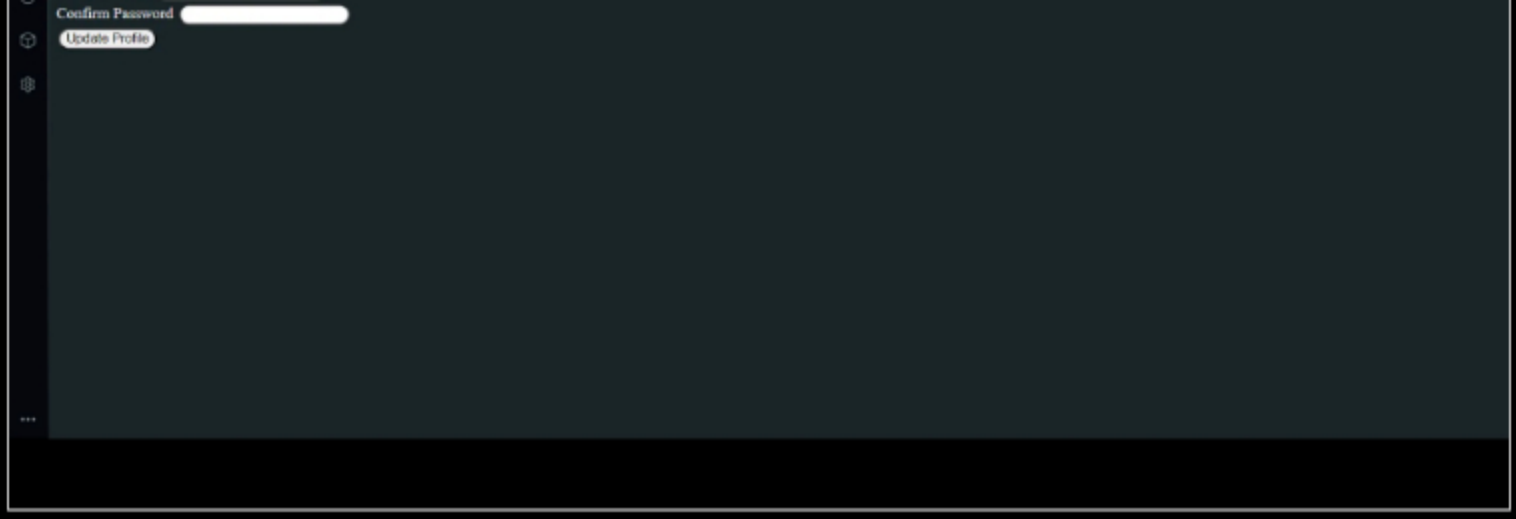


Shows success message of password being updated, "password reset."

Checklist Items (1)

#5 Demonstrate the success message of updating password

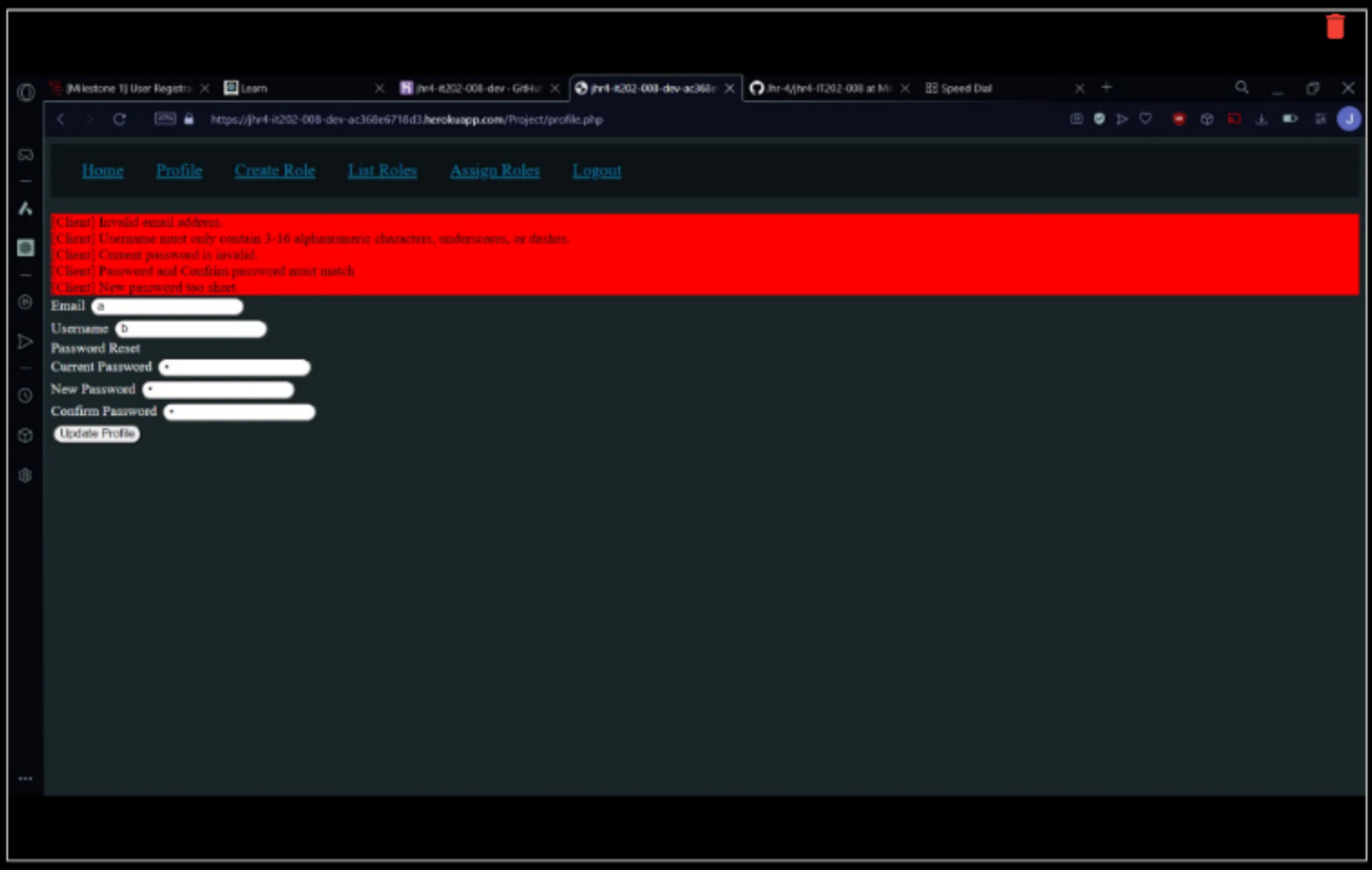




Screenshot showing JS validation if all fields are empty. (The current password is filled in because at least one of the password fields must be filled in to show validation of __ password must be filled. If this isn't done it would show not filled in even if the user didn't try to password reset.)

Checklist Items (1)

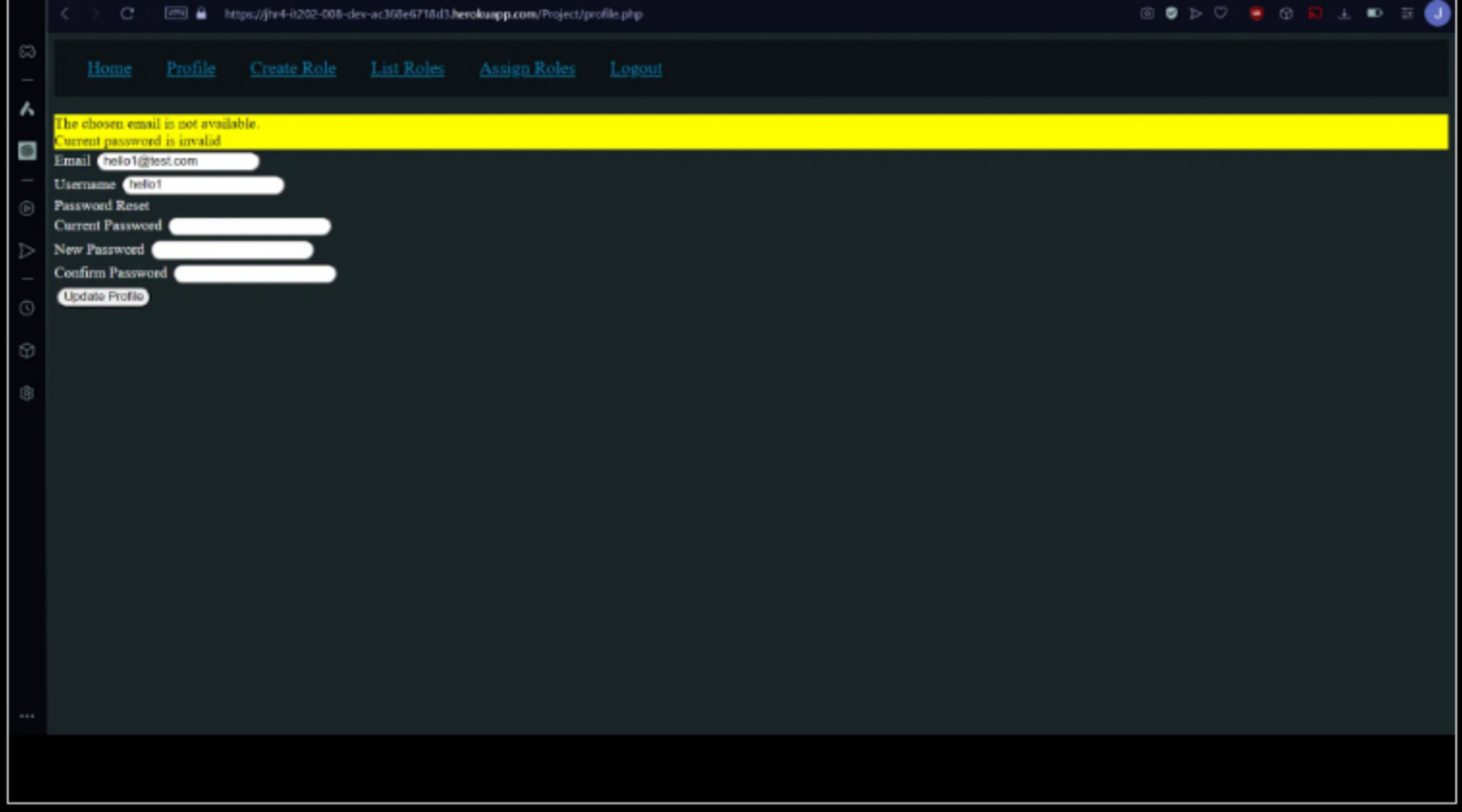
#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)



Shows JS format validation for all fields. Invalid password just means too short (user should know what's wrong).

Checklist Items (1)

#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)



Shows email is not available/already in use. Shows current password is invalid/doesn't match what's in DB.

Checklist Items (1)

#7 Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)

COLLAPSE

Task #4 - Points: 1

Text: Explain the logic step-by-step of how the data is checked and saved for the following scenarios

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Updating Username/Email
#2	1	Updating password
#3	1	Don't just show code, translate things to plain English

Response:

Username/Email:
Data is checked for the username/email with HTML, JS, and PHP validation. The basic idea is the same as the registration page where the HTML and JS are client side. HTML validates the email by utilizing type="email" which makes sures the user types in a valid email. Then to enhance the validation JS is used with regex to check for things like if the email has valid characters, a "@" character, a domain after it followed by a . and a top level domain. The username also uses regex to make sure it has only lowercase alphanumeric characters, _, and - and meets the length of 3-16. While this data is prefilled, the JS and PHP still verify that the user didn't try to change it to empty "" and alerts them that it must be filled in. The PHP validation is also the same to the JS and basically in a different langue.
After this format validation comes calling the database and checking for duplicate emails and usernames which is done by a try catch in which PHP/mySQL is used to create a placeholder of the input and then executing an update to

the database table. If it doesn't work it displays the error like a duplicate email or user. The update of the database occurs thru the user id which is already stored in the session.

Password:
For the password there's also HTML, JS and PHP validation. The HTML validation makes sure all the passwords are length of 8. The JS makes sure that if the user changes/inputs into any field of password (current, new, or confirm), it will validate it. Firstly it checks that all the fields are filled in and aren't just empty, "" if so there is a user user friendly message displayed. Furthermore, the length of the new passwords are also checked and it's made sure that the length is at least 8 and the confirm password is matched with the new password to make sure no typo occurred by user. After this the PHP if no errors the users id is used to fetch their current password which is compared to what the user inputs for current password (as a placeholder). Then, if this goes correctly the new password is hashed and updated in the database Users table.

For both if any validation fails or there's an error an user friendly message is displayed (using the created flash function) for the user to try again.

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/Jhr-4/jhr4-IT202-008/pull/25

Misc (1 pt.)

COLLAPSE

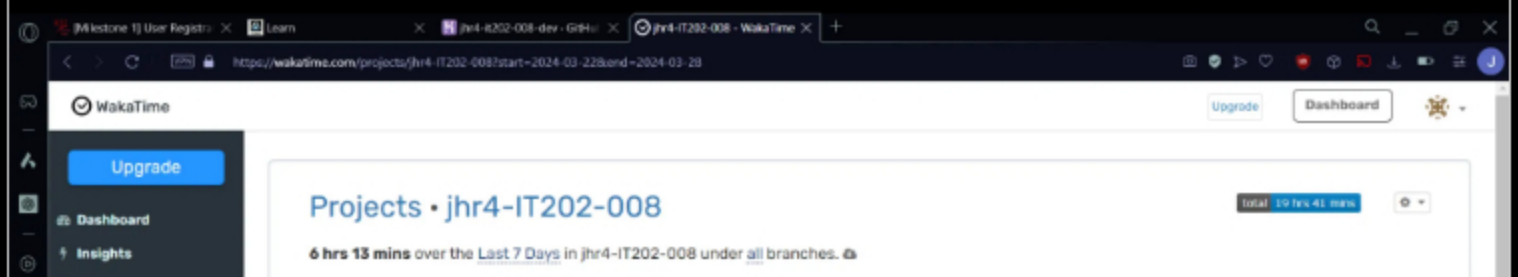
Task #1 - Points: 1

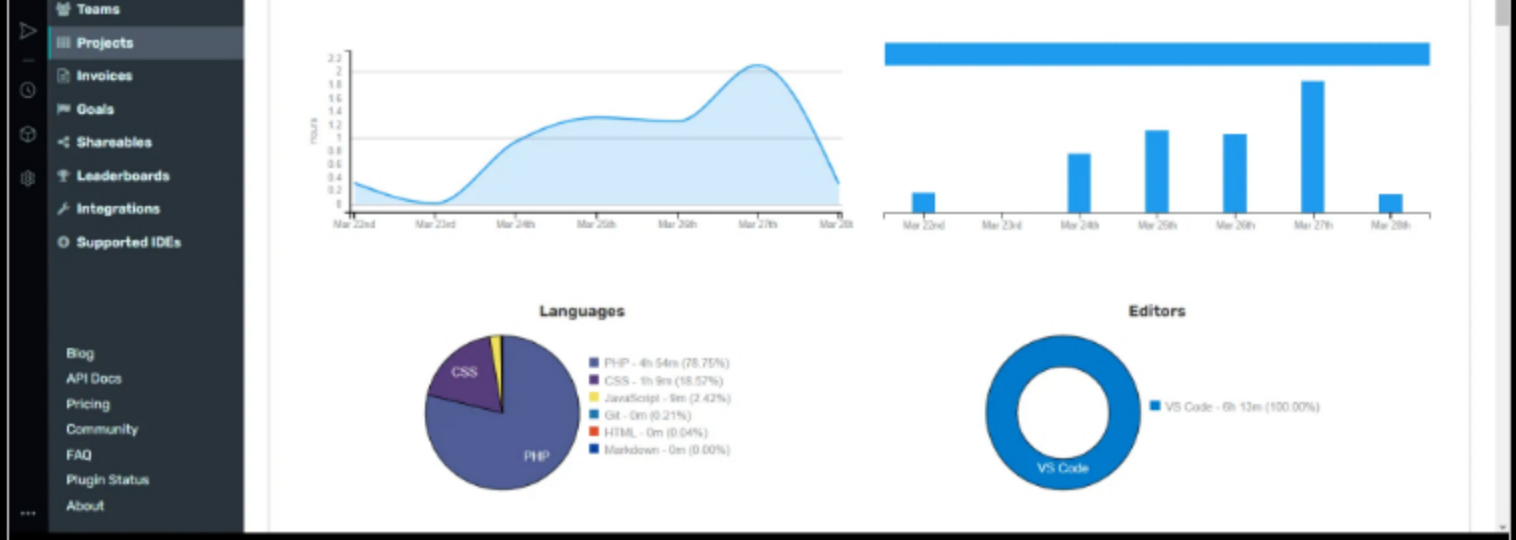
Text: Screenshot of wakatime

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Shows Wakatime of this project.

COLLAPSE

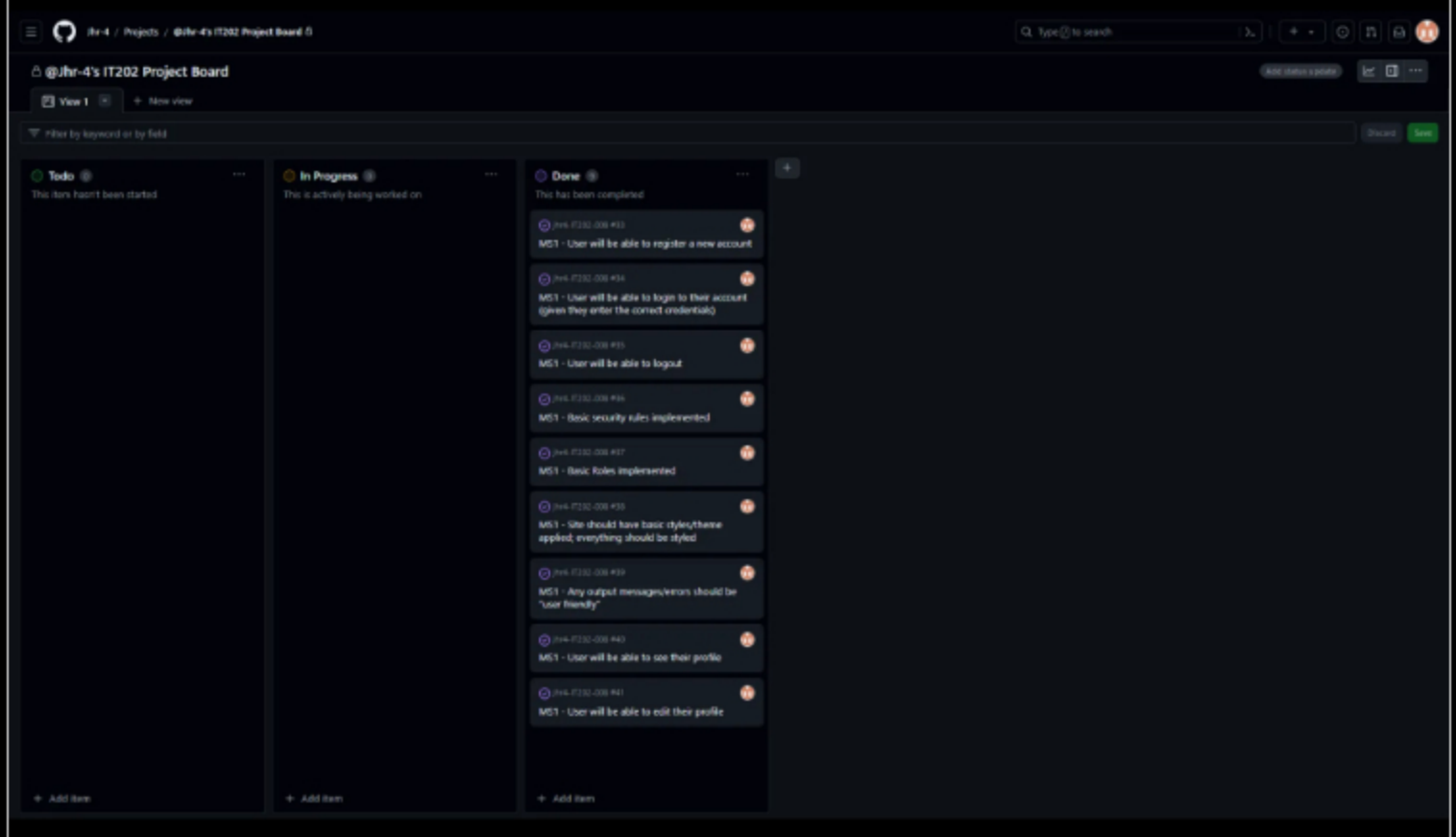
Task #2 - Points: 1

Text: Screenshot of your project board from GitHub (tasks should be in the proper column)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Screenshot of Project Board with 9 issues solved/closed in the done section.

COLLAPSE

Task #3 - Points: 1

Text: Provide a direct link to the project board on GitHub

URL #1

https://github.com/users/Jhr-4/projects/1

COLLAPSE

Task #4 - Points: 1

Text: Provide a direct link to the login page from your prod instance

URL #1

https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/login.php

End of Assignment